PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06K 19/07	A2	(11) International Publication Number: **WO 98/52153** (43) International Publication Date: 19 November 1998 (19.11.98)

(21) International Application Number: PCT/GB98/01411

(22) International Filing Date: 14 May 1998 (14.05.98)

(30) Priority Data:

60/046,514	15 May 1997 (15.05.97)	US
60/046,543	15 May 1997 (15.05.97)	US
09/075,975	11 May 1998 (11.05.98)	US

(71) Applicant: MONDEX INTERNATIONAL LIMITED [GB/GB]; 47-53 Cannon Street, London EC4M 5SQ (GB).

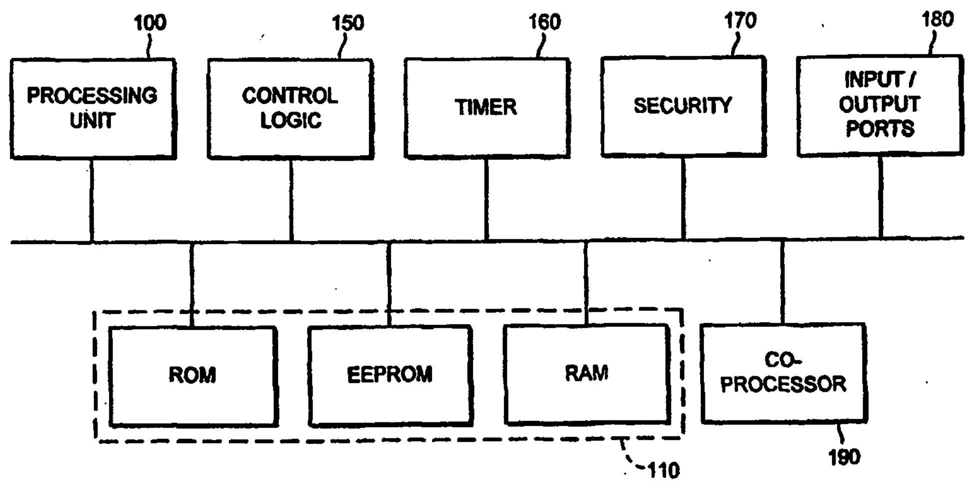
(72) Inventor: RICHARDS, Timothy, Philip; 32 Craig Mount, Radlett, Herts WD7 7LW (GB).

(74) Agent: POTTER, Julian, Mark; D. Young & Co., 21 New Fetter Lane, London EC4A 1DA (GB).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, GW, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: IC CARD WITH SHELL FEATURE



(57) Abstract

There is provided an integrated circuit card having an associated operating mode. The integrated circuit card includes: a microprocessor; a memory coupled to the microprocessor; data stored in the memory representative of the operating mode; an operating system stored in the memory for processing selected information in a first IC card format; a shell application stored in the memory for processing the selected information in a second IC card format; and means for routing the selected information to either the operating system or the shell application responsive to the operating mode. The selected information may be a command, such as a file access command.

IC CARD WITH SHELL FEATURE

BACKGROUND OF INVENTION

Integrated circuit (IC) cards are becoming increasingly used for many different purposes in the world today, principally because they are ideal tools for the delivery of distributed, secure information processing at a low cost. An IC card, also called a "smart card," is a card typically the size of a conventional credit card, but which contains a computer chip on the card. The computer chip on the IC card typically includes a microprocessor, read-only-memory (ROM), electrically erasable programmable read-only-memory (EEPROM), a random access memory (RAM), an input/output (I/O) mechanism, and other circuitry to support the microprocessor in its operations. The computer chip can execute one or more applications stored on the card. Examples of applications that IC cards are being used to store and execute include credit/debit, electronic money/purse, telephone calling card, and loyalty reward applications.

As the use and application of IC cards has increased, IC card standards have been promulgated. For example, the International Organization for Standardization (ISO) and the International Engineering Consortium (IEC) have promulgated several industry-wide standards for IC cards, ISO/IEC 7816-1 through ISO 7816-8. The ISO/IEC standards provide, for example, general guidelines for file structures and referencing methods so that various applications and IC card operating systems can understand one another and work in a cohesive manner. Additionally, in the field of payment systems (such as credit and debit card systems), the EMV '96 Integrated Circuit Card Specification for Payment Systems, Version 3.0, June 30, 1996, available from MasterCard International Incorporated®, specifies file structures and file referencing methods that are generally compliant with ISO/IEC standards 7816-4 and 7816-5. Nonetheless, proprietary IC card standards exist that are not compliant with ISO/IEC standards.

The existence of multiple IC card standards is problematic to the IC card manufacturer, who is required to produce different versions of its IC cards, with different operating systems that are compatible with the different standards. Moreover, since operating systems are typically loaded into the ROM of an IC card when it is initially produced, each time a standard is updated or a new standard is adopted, an IC card manufacturer may be required to distribute new IC cards with an updated operating system compatible with the new or updated standard.

It would advantageous to the card manufacturer, card issuer, application provider, and card user if the operating system of an IC card was not required to be updated each time a new or updated IC card standard was promulgated. These and other technical problems are addressed by embodiments of the present invention.

SUMMARY OF THE INVENTION

The present invention addresses the aforementioned technical problems by introducing a "shell" application that executes "on top" of the operating system and that handles the implementation of IC card standards that are not compatible with the initially loaded operating system of the IC card. Advantageously, the shell application supplements the IC card standards with which the IC card is compatible. Thus, as standards change or new standards are adopted, an IC card needs to be updated only with a new shell application, rather than having to be updated with a new operating system.

According to a preferred embodiment of the present invention, there is provided an integrated circuit card having an associated operating mode. The integrated circuit card includes: a microprocessor; a memory coupled to the microprocessor; data stored in the memory representative of the operating mode; an operating system stored in the memory for processing selected information in a first IC card format; a shell application stored in the memory for processing the selected information in a second IC card format; and means for routing the selected information to either the operating system or the shell application responsive to the operating mode. The selected information may be a command, such as a file access command. In addition, the selected information may be associated with a file structure format.

In accordance with a further preferred embodiment of the present invention, there is also provided a method of loading an application onto an IC card, wherein the application has an associated file mode type and the IC card has an associated operating mode. The method includes the steps of determining whether the file mode type of the application is a predetermined file mode type, and changing the operating mode of the IC card if the file mode type corresponds to the predetermined file mode type. The predetermined file mode type is, for example, a "shell" file mode type, and the operating mode of the IC card is, for example, either "OS" or "shell." Thus, when an application has an associated file mode type of "shell," the operating mode of the IC card is changed from "OS" to "shell."

Preferably, a shell application is not loaded unless it is the first application loaded. In this way, operability of the non-shell applications loaded onto the IC card may be guaranteed. Thus, the method of loading an application according to a further embodiment of the present invention preferably further includes the steps of: determining whether any other applications have already been loaded onto the IC card; loading the application onto the IC card if the file mode type of the application corresponds to the predetermined file mode type and no other applications have already been loaded onto the IC card; and changing the operating mode of the IC card if the file mode type corresponds to the predetermined file mode type and no other applications have already been loaded onto the IC card.

In accordance with another preferred embodiment of the present invention, there is also provided a method of routing a command by an operating system of an IC card, wherein the IC card has an associated operating mode. The method includes the steps of determining whether the operating mode of the IC card is a predetermined operating mode; and routing the command directly to an application if the operating mode of the IC card corresponds to the predetermined operating mode. For example, assuming a SELECT FILE command is received by an IC card from a terminal and the IC card has a shell application loaded thereon, if the operating mode of the IC card and the predetermined operating mode are both "shell," the operating system would route the SELECT FILE command to the shell application.

Preferably, the method of routing further includes the steps of: if the operating mode of the IC card does not correspond to the predetermined operating mode, determining whether the command is a select file command supported by the operating system; and routing the command to an operating system routine responsible for the select file command if the command is a select file command supported by the operating system.

Preferably, the IC card further comprises a currently selected file having an associated file type and the method of routing further comprises the steps of: if the operating mode of the IC card does not correspond to the predetermined operating mode, determining whether the file type of the currently selected file is supported by the operating system; and routing the command to an operating system routine responsible for the file type if the file type of the currently selected file is supported by the operating system. If the file type of the currently selected file is not supported by operating system, the method further comprises the step of routing the command to an application.

In accordance with another preferred embodiment of the present invention, there is also provided a method of delegating control between applications by an operating system of an IC card, wherein the IC card is for use with a defined IC card format and has an associated operating mode. The method includes the steps of storing a shell application in the IC card for communicating with the operating system and for processing information in a format compliant with the defined IC card format; receiving a request by the operating system from a first application for delegating control to a second application; determining whether the operating mode of the IC card is a predetermined operating mode; determining whether the second application corresponds to the shell application; and failing the request for delegating control if the operating mode of the IC card corresponds to the predetermined operating mode and the second application corresponds to the shell application.

In accordance with another preferred embodiment of the present invention, there is also provided a method of initiating communication between an IC card and a terminal, wherein the IC card comprises a microprocessor and a memory, the memory having stored therein an operating system, a shell application, and data representative of an operating mode of the IC card, the operating mode representing whether selected information is to be routed to the operating system or the shell application. The method of initiating includes the steps of receiving a reset signal by the IC card from the terminal; and returning an answer-to-reset from the IC card to the terminal based on the operating mode of the IC card.

Preferably, a plurality of answer-to-reset files are stored in the memory of the IC card, and the step of returning an answer-to-reset comprises selecting one of the answer-to-reset files based on the operating mode. The selected information may be a command, such as a file access command. In addition, the selected information may be associated with a file structure format.

BRIEF DESCRIPTION OF THE DRAWINGS

Preferred embodiments in accordance with the invention will now be described by way of example only, with reference to the accompanying drawings, in which:

Fig. 1 is a schematic representation of an IC card in accordance with a preferred embodiment of the present invention;

Fig. 2 is a perspective view of an IC card and terminal in accordance with a preferred embodiment of the present invention;

Fig. 3 is a functional block diagram of an IC card in accordance with a preferred embodiment of the present invention;

Fig. 4 is an exemplary hierarchical file structure according to the EMV Specification;

Figs. 5A and 5B are flowcharts illustrating the steps for a load_file command used in accordance with a preferred embodiment of the present invention;

Fig. 6 is a flowchart illustrating the steps for a delete_file command used in accordance with a preferred embodiment of the present invention;

Fig. 7 is a flowchart illustrating the steps for a route command used in accordance with a preferred embodiment of the present invention;

Fig. 8 is a flowchart illustrating the steps for a delegate_request command used in accordance with a preferred embodiment of the present invention; and

Fig. 9 is a flowchart illustrating the steps for a determine_ATR_status command used in accordance with a preferred embodiment of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

Fig. 1 provides a schematic representation of a typical IC card 10 that can be used with the presently claimed invention. The IC card 10 includes an integrated circuit 12 and one or more electrical contacts 14, connected to the integrated circuit 12, for communication between the integrated circuit 12 and devices outside the IC card 10.

Fig. 2 shows an example of a device with which the IC card 10 communicates. As used in this specification and the appended claims, the term "terminal" shall be used to generically describe devices with which an IC card may communicate. A typical terminal 20, as shown in Fig. 2, includes a card reader 22, a keypad 24, and a display 26. The keypad 24 and the display 26 allow a user of the IC card 10 to interact with the terminal. The keypad 24 allows the user to select a transaction, to enter a personal identification number ("PIN"), and to enter transactional information. The display 26 allows the user to receive informational messages and prompts for data entry. Other types of terminals may include IC card compatible ATM machines and telephones.

Fig. 3 provides a functional block diagram of the integrated circuit 12. At a minimum, the integrated circuit 12 includes a processing unit 100 and a memory unit 110. Preferably, the integrated circuit 12 also includes control logic 150, a timer 160, security circuitry 170, input/output ports 180, and a co-processor 190. The control logic 150 provides, in conjunction with the processing unit 100, the control necessary to handle communications between the memory unit 110 and input/output ports 180. The timer 160 provides a timing reference signal for the processing unit 100 and the control logic 150. The security circuitry 170 preferably provides fusible links that connect the input/output ports 180 to internal circuitry for testing during manufacturing. The fusible links are burned after completion of testing to limit later access to sensitive circuit areas. The co-processor 190 provides the ability to perform complex computations in real time, such as those required by cryptographic algorithms.

The memory unit 110 may include different types of memory, such as volatile and non-volatile memory and read-only and programmable memory. For example, as shown in Fig. 3, the memory unit 110 may include read-only memory (ROM), electrically erasable programmable read-only memory (EEPROM), and random-access memory (RAM).

The memory unit 110 stores IC card data such as secret cryptographic keys and a user PIN. The secret cryptographic keys may be any type of well-known cryptographic keys, such as the private keys of public-key pairs. Preferably, the secret cryptographic keys are stored in a secure area of ROM or EEPROM that is either not accessible or has very limited accessibility from outside the IC card.

The memory unit 110 also stores the operating system of the IC card. The operating system loads and executes IC card applications and provides file management and other basic card services to the IC card applications. Preferably, the operating system is stored in ROM.

In addition to the basic services provided by the operating system, the memory unit 110 may also include one or more IC card applications. For example, if the IC card is to be used as an electronic cash card, an application called MONDEX™ PURSE might be included on the IC card, which loads an electronic value of a certain currency from a user's account in a financial institution onto the IC card. An application may include both program and data files, which may be stored in either ROM or EEPROM.

To enable the inter-operability of different terminals with different IC cards and applications, standards have been promulgated with respect to the organization of files stored on an IC card. For example, in the payment systems industry, the EMV '96 Integrated Circuit Card Specification for Payment Systems, Version 3.0, June 30, 1996, available from MasterCard International Incorporated® (hereinafter the "EMV Specification"), incorporated herein by reference in its entirety, sets forth a hierarchical tree structure for accessing files, which is generally compliant with the ISO/IEC 7816-4 and 7816-5 standards. An illustrative example of such a hierarchical tree structure is provided in Fig. 4.

In Fig. 4, there are shown four types of file categories: the Directory Definition File (DDF), the Directory File (DIR), the Application Definition File (ADF), and the Application Elementary File (AEF). According to the EMV Specification, each DDF contains one DIR. Each DIR may contain one or more ADF and/or DDF. Each ADF contains one or more AEF, which are files containing data related to a particular application.

According to the EMV Specification, files are referenced either by a unique name or by a short file identifier (SFI). A DDF or ADF is referenced by its unique name using a SELECT command. Once a particular DDF or ADF is selected, a corresponding DIR or AEF is referenced with an SFI using a READ RECORD command. In the case of a DIR, the SFI is in the range of 1 to 10. In the case of an AEF, the SFI is in the range 1 to 30. The EMV Specification sets forth at least one mandatory DDF with a unique name of "1PAY.SYS.DDF01."

The format for a SELECT command for selecting a DDF or ADF according to the EMV Specification is shown in Table 1. In response to a SELECT command for a DDF, an IC card returns the SFI of the DIR attached to the DDF. When an ADF is selected, an IC card returns information that the terminal may use, in conjunction with other commands, to retrieve the SFI of AEFs related to the ADF.

Once the SFI of a DIR or AEF is known, a terminal may use the READ RECORD command to read the records of the DIR or AEF. The format of the READ RECORD command according to the EMV Specification is shown in Table 2.

TABLE 1: SELECT Command Format

Byte Number	Value
1	Hexadecimal “00”
2	Hexadecimal “A4”
3	Hexadecimal “04”
4	Hexadecimal “00”
5	Length of File Name (Hexadecimal “05” - “10”)
6-21	File Name (number of bytes variable depending on length of file name)
Last	Hexadecimal “00”

TABLE 2: READ RECORD Command Format

Byte Number	Value
1	Hexadecimal “00”
2	Hexadecimal “B2”
3	Record Number
4	SFI
5	Hexadecimal “00”

Although the EMV Specification sets a standard for file organization within the payment systems industry, other IC card file organization standards may exist in other industries. Some may be proprietary and may not be generally compatible with the EMV Specification or ISO/IEC 7816-4 or 7816-5.

Typically, an IC manufacturer who desires to produce IC cards compatible with the EMV Specification and other proprietary specifications must produce IC cards with different operating systems to implement the different file structures and different file referencing and access methods defined by the various specifications. According to embodiments of the presently claimed invention, however, a manufacturer may produce an IC card with a single operating system and execute different shell applications to implement the different standards.

Figs. 5A to 9 are flowcharts illustrating a preferred embodiment of IC card operating system routines capable of supporting a shell application. In the embodiment of Figs. 5A to 9, the operating system is a multiple application operating system that runs on IC cards, such as the MULTOS™ operating system from Mondex International Limited. Such an operating system includes routines for loading and deleting applications, routines for routing commands to appropriate operating system processes or applications, routines for handling delegation of processing between applications, and routines for handling the answer-to-reset (ATR) message.

In the embodiment of Figs. 5A to 9, only one shell application can be loaded onto an IC card at any one time. Once the shell application is loaded, it is valid for all applications loaded on the IC card. Preferably, the operating system has a delegation feature, such as the delegation feature described in the United States patent application entitled "Multi-Application IC Card with Delegation

Feature," by Everett et al., filed April 23, 1998, which is hereby incorporated by reference to Annex A attached hereto. When the shell application receives a command from the operating system, it interprets the command and/or delegates control to the application associated with the command. If control is delegated to an application, when the application is finished, it returns control to the shell application. The shell application then returns any response to the operating system in the proper format for transmission to the terminal.

Although for the sake of simplicity the preferred embodiment loads only a single shell application at a time, the present invention is not limited to such an embodiment. It is within the scope of embodiments of the present invention for multiple shell applications to be loaded onto an IC card and to be used with different sets of applications.

As a matter of notation, the data elements referred to in the flowcharts of Figs. 5A to 9 follow a dot notation convention where the data element following the dot (".") is a component of the data element preceding the dot. For example, the data element *file_mode* includes two components: *file_mode_type* and *application_id*. In the dot notation used, the first component data element is referred to as *file_mode.file_mode_type* and the second component data element is referred to as *file_mode.application_id.*

Figs. 5A and 5B are flowcharts illustrating the implementation of a file loading routine by an operating system capable of supporting a shell application. In step 510, the routine receives the file loading command *load_file_command* from the security manager of the operating system, *OS_Security_Manager*. In step 520, after receiving the command, the routine checks whether the application identification number associated with the command, *load_file_command.application_id*, is present in the operating system control information, *os_control_info.application_id*. If the application identification number is already present, in step 521, the routine sets the response status *load_file_response.status* to "failed" and sets the error description *load_file_response.error_cause* to "duplicate application id." This error response indicates that the application is already loaded and cannot be loaded again. The error response *load_file_response* is then returned to the *OS_Security_Manager*.

If the application identification number of the application to be loaded is not present, in step 530, the routine checks the file mode type of *load_file_command*. The file mode type may be, for example, "shell" or "non-shell." A "shell" file mode type indicates that the application to be loaded is a shell application, while a "non-shell" file mode type indicates that the application to be loaded is not a shell application.

If the application to be loaded is a shell application, the routine further checks whether *os_control_info* is empty. If *os_control_info* is not empty, then one or more applications have already been loaded onto the IC card. If this is the case, in step 531, the routine sets the response status *load_file_response.status* to "failed" and sets the error description *load_file_response.error_cause* to "application already loaded." This error response is a result of the restriction that the shell application is to be valid for all applications loaded onto the IC card. To ensure that all applications will operate correctly with the shell application, the shell application must be the first application loaded onto the IC card.

Assuming that an error condition has not been triggered in steps 520 and 530, the directory file and *os_control_info* are updated with the appropriate application information in steps 540 and 550.

With reference to Fig. 5B, in step 560, the file mode type of *load_file_command* is checked once again. If the file mode type is "shell," then in step 570, the *file_mode* and the *selected_file* data elements are updated. The *file_mode* data element contains both the *file_mode_type* of the IC card and the *application_id* of the shell application. The *file_mode.file_mode_type* variable represents the operating mode of the IC card and, thus, may also be referred to as the "operating mode." The operating mode of the IC card may be, for example, either "OS" or "shell." "OS" mode indicates that a shell is not loaded, while "shell" mode indicates that a shell is loaded. The *selected_file* data element contains the *application_id* and the *file_type* of the currently selected file.

In step 570, *file_mode.file_mode_type* is set to "shell." The *file_mode.file_mode_type* represents the operating mode of the IC card and, thus, is also referred to as the "operating mode." In addition, the application identification number of the currently selected file is set to the application identification number of the shell application. The *file_type* of the selected file is set to "dedicated file," indicating that file commands are not to be handled by the operating system.

In step 580, the response status *load_file_response.status* is set to "success" and is returned to the *OS_Security_Manager*.

Fig. 6 is a flowchart illustrating the implementation of a file deleting routine by an operating system capable of supporting a shell application. In step 610, a *delete_file_command* is received from the *OS_Security_Manager*. In step 620, checking is performed to verify that the application being deleted exists in *os_control_info* —— i.e., that the application is loaded on the IC card. If the application identification number is not in *os_control_info*, then in step 670, the response status *delete_file_response.status* is set to "failed" and the error description *delete_file_response.error_cause* is set to "application not loaded."

If the application is loaded on the IC card, in step 630 checking is performed to determine whether the file mode type of the application being deleted, *delete_file_command.file_mode_type,* is equal to "shell." Checking is also performed to determine whether the application identification number of the application being deleted, *delete_file_command.application_id,* is equal to the application identification number assigned to the file mode of the IC card, *file_mode.application_id.* In short, checking is performed to determine whether a loaded shell application is being deleted.

If a loaded shell application is being deleted, in step 680, *file_mode.file_mode_type* is set to "OS" and *selected_file.file_type* is set to the default file type for the IC card, i.e., "master file."

In step 640, the directory file record corresponding to the application is deleted from the directory in which it is stored. In step 650, the application identification number of the application is deleted from *os_control_info*. In step 660, *delete_file_response.status* is set to "success" and the response status is returned to the *OS_Security_Manager*.

Fig. 7 is a flowchart illustrating the implementation of a command routing routine by an operating system capable of supporting a shell application. In step 710, the route routine receives a command from the cardholder — i.e., a command from outside of the IC card. In step 720, checking is performed to determine the operating mode of the IC card. If *file_mode.file_mode_type* is not equal to "OS," a shell application has been loaded onto the IC card. Thus, the command from the cardholder is sent directly to the currently selected application or applications. In the typical case, the currently selected application will be the shell application. It may be the case, however, that the shell application has delegated control to another application and that that application receives and processes the command directly.

If the operating mode of the IC card is equal to "OS," the various conditions defined in steps 730 to 750 are checked. In step 730, if the command is a *select_file* command, the command is sent to the *select_file* routine of the operating system. In step 740, if the file type of the currently selected file is "master file," the command is sent to the *provide_card_facilities* routine of the operating system, which handles commands associated with the master file type. Similarly, in step 750, if the file type is "directory file," the command is sent to the *read_card-level_data_files* routine of the operating system, which handles commands associated with the directory file type. If none of the conditions in steps 730 to 750 are satisfied, then the selected file must be an application. Therefore, the command is sent to the currently selected applications.

Fig. 8 is a flowchart illustrating a delegate request checking routine that is necessary if an operating system supports both a shell application and a delegate feature. In step 810, a *delegate_request* is received from an application. In step 820, checking is performed to determine whether the operating mode of the IC card is "shell" and whether the application identification number of the delegated application (the application to which control is being sought to be transferred) is the same as the application identification of the shell application of the IC card. If both conditions are true, then an application is attempting to delegate control to the shell application. Since the shell application is the first application loaded and selected, and thus delegates control to all other applications, such a delegation would be recursive. Recursive delegation is not allowed. In step 830, therefore, *delegate_response.status* is set to "failed" and *delegate_response.error_cause* is set to "recursive shell delegation." The delegate response is returned to the delegator applications. In step 820, if it is determined that the delegator application has submitted a proper, non-recursive delegate request, the request is processed in accordance with the operating system's delegate handling procedures.

When an IC card is inserted into a terminal, it receives a reset signal. To initiate communication with the terminal, the IC card must respond to the reset signal with an appropriate answer-to-reset (ATR) message. Fig. 9 is a flowchart illustrating an ATR routine for an IC card operating system that supports a shell application.

In step 910, the operating mode of the IC card is checked. If the *file_mode.file_mode_type* is equal to "OS," in step 920, the file type of *selected_file* is set to the default "master file" and *s_ATR_status* is set to "default ATR." Otherwise, if the operating mode of the IC card is "shell," in step 930, the file type and application identification number of the selected file are set to "dedicated file" and *file_mode.application_id*, respectively. *s_ATR_status* is set to "shell ATR." In both cases, *s_ATR_status* is returned to the *control_ATR* routine of the operating system. Using *s_ATR_status*, the *control_ATR* routine responds with the appropriate ATR to the reset signal from the terminal. The appropriate ATR may be stored in different files on the IC card, which are selected based on *s_ATR_status*.

Although the present invention has been described with reference to certain preferred embodiments, various modifications, alterations, and substitutions will be known or obvious to those skilled in the art without departing from the spirit and scope of the invention, as defined by the appended claims.

The scope of the present disclosure includes any novel feature or combination of features disclosed therein either explicitly or implicitly or any generalisation thereof irrespective of whether or not it relates to the claimed invention or mitigates any or all of the problems addressed by the present invention.

The application hereby gives notice that new claims may be formulated to such features during the prosecution of this application or of any such further application derived therefrom. In particular, with reference to the appended claims, features from dependant claims may be combined with those of the independent claims in any appropriate manner and not merely in the specific combinations enumerated in the claims.

ANNEX A TO THE DESCRIPTION

ANNEX A

MULTI-APPLICATION IC CARD WITH DELEGATION FEATURE

ANNEX A TO THE DESCRIPTION

BACKGROUND OF INVENTION

Integrated circuit ("IC") cards are becoming increasingly used for many different purposes in the world today. An IC card (also called a smart card) typically is the size of a conventional credit card which contains a computer chip including a microprocessor, read-only-memory (ROM), electrically erasable programmable read-only-memory (EEPROM), a random access memory (RAM), an Input/Output (I/O) mechanism and other circuitry to support the microprocessor in its operations. An IC card may contain a single application or may contain multiple independent applications in its memory. MULTOS™ is a multiple application operating system which runs on IC cards, among other platforms, and allows multiple applications to be executed on the card itself. The multiple application operating system present on the IC card allows a card user to run many programs stored in the card (for example, credit/debit, electronic money/purse and/or loyalty applications) irrespective of the type of terminal (i.e., ATM, telephone and/or POS) in which the card is inserted for use.

A conventional single application IC card, such as a telephone card or an electronic cash card, is loaded with a single application card and only executes that one application when inserted into a terminal. For example, a telephone card could only be used to charge a telephone call and could not be used as a credit/debit card. If a card user desires a variety of application functions to be performed by single application IC cards issued to him or her, such as both an electronic purse and a credit/debit function, the card user would be required to carry

ANNEX A TO THE DESCRIPTION

multiple physical cards on his or her person, which would be quite cumbersome and inconvenient. If an application developer or card user desired two different applications to interact or exchange data with each other, such as a purse application interacting with a frequent flyer loyalty application, the card user would be forced to swap multiple cards in and out of the card-receiving terminal during the transaction, making the transaction difficult, lengthy and inconvenient.

Therefore, it is beneficial to store multiple applications on the same IC card. For example, a card user may have both a purse application and a credit/debit application on the same card so that the user could select which type of payment (by electronic cash or credit card) to use to make a purchase. Multiple applications could be provided to an IC card if sufficient memory exists and an operating system capable of supporting multiple applications is present on the card.

The increased flexibility and power of storing multiple applications on a single card create new challenges to be overcome concerning the integrity and security of the information (including application code and associated data) exchanged between the individual card and the application provider as well as within the entire system when communicating information between applications.

For instance, the existence of multiple applications on the same card allows for the exchange of data between two applications, while one of the applications is being executed. As stated above, a frequent flyer loyalty program may need to be accessed during the execution of an electronic purse application. If data is passed between applications in an insecure manner, it may be possible for a third party monitoring the transaction to determine the contents of the transferred

ANNEX A TO THE DESCRIPTION

data or even other private data associated with one or both of the applications. Thus, it would be beneficial to provide an application architecture and memory organization which protects an application's data from being discovered by a third party when it is exchanged with other applications present on the IC card.

Accordingly, it is an object of the invention to provide an application architecture and memory organization which provides for a secure data interaction between applications and allows multiple applications to be accessed while performing a desired task or function.

SUMMARY OF THE INVENTION

The present invention provides for a multiple application architecture for an IC card called an application abstract machine (AAM) and a method for implementing that architecture. The processing of multiple applications is accomplished by generating for at least one application (the "first application") a data memory space including at least two segments, a volatile memory segment and a non-volatile memory segment, commencing the execution of the first application's instructions; delegating or switching execution from the first application to the delegated application and in so doing, saving any data generated by the first application in the logical data memory space associated with the first application; executing the second application's instructions; retrieving the saved data and completing with this data the execution of the first application's instructions.

ANNEX A TO THE DESCRIPTION

Additional delegation commands can be issued by the second application or other subsequent applications. The command delegated is interpreted by a delegated application in the same manner as a selection command being issued directly by a terminal and therefore each application performs the security functions at the same level as if a terminal is issuing the command.

The volatile memory segment can further be separated into public ("Public") and dynamic ("Dynamic") portions. Data can be exchanged between a plurality of applications and/or a terminal when stored in the Public region of the data memory. The Dynamic memory region can be used solely as temporary work space for the specific application being executed.

BRIEF DESCRIPTION OF THE DRAWINGS

Further objects, features and advantages of the invention will become apparent from the following detailed description taken in conjunction with the accompanying figures showing illustrative embodiments of the invention, in which

Fig. 1 is block diagram illustrating the data memory space segment and associated registers for an IC card application using the AAM organization;

Fig. 2 is a block diagram illustrating the code memory and the data memory spaces for an IC card application using the AAM architecture;

Fig. 3 is a flow diagram illustrating the steps of performing a request for a delegation function by one application to another;

Fig. 4 is a flow diagram illustrating the steps of performing a return

ANNEX A TO THE DESCRIPTION

delegation control function for a delegate application to a delegator application;

Fig. 5 is a flow diagram illustrating the steps of performing an inquire delegator ID request of a delegation function;

Fig. 6 is a block diagram of an IC card chip which can be used as a platform in accordance with the invention; and

Figures 7A, 7B and 7C illustrate multiple delegation calls made between three applications.

Throughout the figures, the same reference numerals and characters, unless otherwise stated, are used to denote like features, elements, components or portions of the illustrated embodiments. Moreover, while the subject invention will now be described in detail with reference to the figures, it is done so in connection with the illustrative embodiments. It is intended that changes and modifications can be made to the described embodiments without departing from the true scope and spirit of the subject invention as defined by the appended claims.

ANNEX A TO THE DESCRIPTION

DETAILED DESCRIPTION OF THE INVENTION

The present invention provides for a method and apparatus for processing multiple application programs with associated data stored on an IC card which can be accessed and executed. An application stored on the card can be selected by a terminal, or other interface device, or another application. Each application program which is stored on the IC card when executed is allocated a memory space organized by the program's software code (instructions which are executed by a processor located on the IC card) and the associated data which the application stores and uses during execution of the program.

For example, a multi-application card may store a purse application, or an electronic money application, and a specific loyalty application such as a frequent flyer awards application. Each application has software code and associated data to support the execution of that software code. Each application is allocated a memory space when executed. In this example, there is interaction between the two applications stored on the card. For each dollar electronically spent to make a purchase, the user may be entitled to one frequent flyer mile which is stored and processed by the frequent flyer program. The purse application need not be aware of the specific loyalty program stored on the card, but instead may contain an instruction to communicate with any loyalty program stored on the card. The loyalty program will require input data representative of the amount of a particular electronic value so that it can update its own stored data of current frequent flyer miles for the user of the card.

ANNEX A TO THE DESCRIPTION

When two applications need to communicate during the same transaction, a system architecture is required to process both applications in an efficient and secure manner. One approach could be a windows type model where both applications could be running at the same time. Presently, however, IC card platforms are not powerful enough to simultaneously operate multiple programs efficiently. Also, transferred data may be exposed to unwanted third party access. The solution to this problem, provided by the current invention, which is described in greater detail below, is to selectively interrupt the execution of applications in a secure manner. This allows the integrity of the applications' data to be maintained and allows the best utilization of the available memory space in the IC card.

An efficient architecture for processing multi applications in an IC card is termed an Application Abstract Machine (AAM) architecture and is described herein. The AAM Architecture applies to any platform independent of its hardware and enables developers to write applications to store on the IC cards which are portable across many different types of platforms (e.g., IC cards built by different manufacturers with different processor configurations) without the need for knowledge about the specific hardware of the platform.

An application abstract machine (AAM), a term for the memory allocation and organization for the data stored and used by each application, is created for each application stored on the IC card which is executed by the processor on the card. In order to ensure data integrity and security when data is transferred between applications which are executed on the IC card, only one application on the IC card is allowed to be executed at a time. Each application has

ANNEX A TO THE DESCRIPTION

a data memory space which is virtually allocated and mapped onto the physical memory addresses available in the IC card memories. Data is then passed between two or more applications within a specified memory location and in a manner consistent with transferring data to an external terminal or device with which the IC card is securely interacting. At a general level, each AAM space created for each application being executed includes two separate address spaces, one for the program code itself and one for the program data which is stored and/or used by the application. The program data address space is effectively divided into three segments: a Static segment, a Dynamic segment and a Public segment which are described in more detail in conjunction with Figure 1. As stated above, the Static, Dynamic and Public segments are logically mapped to the physical memory; they are virtual memory segments as opposed to physical memory segments. The AAM data address space is preferably addressed and processed using seven different address registers and two control registers.

Figure 1 shows an illustrative diagram of a logical data space allocation 101 created for an application used in conjunction with the present invention. The AAM data portion 101 includes a Static data space 103, a Public data space 105 and a Dynamic data space 107. Also shown are a series of address registers: the Static base address register 109, the Static top address register 111, the Public base address register 113, the Public top address register 115, the Dynamic base address register 117, the Dynamic top address register 121 and local base address register 119 which serves as a local stack frame pointer in the Dynamic data space when the application is being executed. The address registers

ANNEX A TO THE DESCRIPTION

can contain physical memory addresses but preferably contain offset addresses for the various data address spaces in order to be hardware independent. An example of the overall address space is 64K bytes, although the size varies with the applicable platform and the available memory size. The registers can also be considered pointers or can be any other conventional addressing mechanism.

Within the allocated AAM data space 101, the Static portion of the memory is non-volatile which is not erased after power is removed from the IC card (such as EEPROM), the Dynamic space is volatile (such as RAM) which may be erased after power is removed from the card and the Public space is also volatile (such as RAM). An IC card can receive power from a terminal after it is interfaced into the terminal. Although an IC card may contain a battery to maintain some power for memory and circuitry, volatile memory will typically be erased after the IC card is removed from its power source.

The defined AAM data space has bytes in each segment which are contiguous, so that applications can perform pointer and offset arithmetic. For example, if the segment addresses "1515" and "1516," or any other pair of sequential numbers, are both valid and are present within the same segment, then they address adjacent bytes. This allows offset values stored in registers to determine the location of a desired memory address. The segment address of the first byte of the Static segment is zero, so that the segment address of a given location within the Static region is equal to its offset.

Pointers to other specific regions of the Static data area can be stored in the Static data because the Static region is non-volatile. For example, if the card

ANNEX A TO THE DESCRIPTION

user's name is stored in the Static memory of a credit/debit application, the application will know the card user's name will always be stored in the 5th memory location above the starting point for the Static portion of memory. The location can be noted as SB[5] or the 5th byte above the Static Bottom. Since the Static memory is non-volatile, it will not be erased after each transaction and the application will always know of its location relative to the Static segments' address registers.

On the other hand, the segment address of any location in the Dynamic or Public segments is not always equal to a particular offset from the beginning of the respective segment because the contents of those segments change for each operation. The fourth location in the Dynamic segment will be different for each operation performed by the application. The address of a memory location of Dynamic or Public segment is fixed preferably only for the duration of one command-response pair operation. Because segment addresses in Dynamic or Public are not fixed, MULTOS Executable Language (MEL)™ instructions (or any other program instructions) cannot refer to data using only segment addresses. Instead, a tagged address preferably is used to identify data which is to be retrieved, manipulated, transferred and/or stored with the IC card system.

A tagged address is a nineteen bit value consisting of a three bit tag (address register number) and a sixteen bit offset. Each of the seven address registers for the AAM data space contain a segment physical address. For instance, the address registers SB 109 and ST 111 point to the boundaries of the Static, the address registers PB 113 and PT 115 point to the boundaries of the Public and the address registers DB 117 and DT 121 point to the boundaries of the Dynamic. For

ANNEX A TO THE DESCRIPTION

each segment, the top register points to the byte immediately after the last valid byte. For example, the last valid byte of the Static is ST[-1]. Register LB functions as a stack frame pointer. It points to a location in the Dynamic segment to indicate a specific byte of local data for the currently executing application.

Referring to Figure 1, the allocated Static segment 103 contains the application's non-volatile data. Static data includes data which is associated with each application for every transaction such as the card user's name, account number, PIN value and address. Static data also includes variable data which is stored for use in future transactions using the application. For example, in a purse transaction, the electronic value data would be read from the Static segment and later saved in the Static segment at the end of the transaction. Additionally, transaction information data or available credit limits in the case of a credit/debit application would be stored in Static data.

The Static data is addressed using register SB (Static Base) and the register ST (Static Top) as offset registers. These registers contain the offset value from a physical address in a memory on the IC card. The individual memory location is then further offset from these starting points such as SB[3] or ST[-5]. SB is defined as zero and ST is equal to the size of the application's Static data which is set when the application is loaded onto the IC card. The multiple application operating system ensures that no other application can read or write the data stored in the Static segment of a particular application. Using current technology, the Static segment is preferably mapped onto an EEPROM (Electrically Erasable Programmable Read-Only Memory) which is non-volatile.

ANNEX A TO THE DESCRIPTION

The Dynamic segment 107 contains the application's volatile or temporary data. Dynamic data includes data which is temporarily used during the execution of an application such as intermediate values used in calculations or working variables. For example, a purse application may temporarily store the value of a transaction in order to reduce the amount of the value in the purse. The temporary data is used much like conventional computer programs use RAM to perform their assigned operations. The Dynamic segment preferably is divided into two parts, the session data portion and the stack data portion. The size of the session data is a constant for each application and is determined when the application is loaded. The stack holds variable data which is unique to the particular transaction being executed. The stack data portion stores data in a last-in-first-out manner. The stack is initially empty, but expands and contracts during execution of the application.

The Dynamic data is addressed from the register DB 117 to register DT 121. Register LB 119 serves as a local stack frame pointer to particular memory locations in the Dynamic segment for delegate commands or function calls. Register LB 119 is used to address the topmost frame, that of the currently executing function's session data. Register DT 121 serves as an address offset for the stack pointer. A one byte data item at the top of the stack is addressed as DT[-1], the next byte below is addressed by DT[-2], and so on. A push operation increments the relative value of DT for each item on the stack and a pop operation decrements the relative value of DT for each item on the stack. For example, a data element located at DT[-5] will be located at DT[-6] after an additional data

ANNEX A TO THE DESCRIPTION

item is placed on the stack.

When an application is being executed, the Dynamic segment created for that application also contains the application's session data which is used in performing the assigned task(s) or operation(s). The multiple application operating system ensures that no other application can read or write the data stored in the Dynamic segment of a particular application. The session data is set to zero upon the start of the execution of the application. Stack data will be saved in the stack if the application delegates a task or operation to another application.

A delegation function occurs when one application selects another application to process a command instead of processing the command itself. An example of a delegation function occurs when a delegator application receives a command that it does not recognize or is not programmed to process. The selected application should not reject the command and provide an error response to the interface device (IFD), but instead should pass the command to the appropriate receiver, or delegated application. In order to perform a delegation, the delegator calls the Delegate primitive. The Delegate primitive is a subroutine recognized by the multiple application operating system which is executed when the operating system interprets the Delegate instruction. Primitives can be stored as part of the operating system itself, loaded as a separate routine when the operating system is installed. Primitives are preferably written in machine executable language so that they can be executed quickly although they could be written in a higher level language. When a Delegate command is executed, execution of the delegating application is suspended, and the delegated application is executed instead. The

ANNEX A TO THE DESCRIPTION

delegated application then generates its own data memory space according to the AAM architecture. The data stored in the Public memory space of the first application (stored in RAM) is sent to the Public memory space of the second application (which could be physically the same memory but is allocated separately for each application) so that data can be passed between the applications. The Dynamic memory space is also shared although data is saved in a stack for the delegator and the other portions initialized before the delegated application is executed because the Dynamic data is secret.

In most cases, the delegated application processes the command exactly as though the command has arrived directly from an interface device. When the delegated application has finished processing the command, and has written a response into the allocated Public memory segment, it exits as normal. The delegator then resumes execution at the instruction address following the executed instruction which called the Delegate primitive. The response generated by the delegated application is retrieved or accessed from the allocated Public memory space. The delegator application may simply exit in turn, thus sending the response to the IFD, or may carry out further processing before exiting.

Another example of a delegation operation occurs when two applications need to share data. If an application A always returns a data item N when processing a command B, then another application which also returns data item N in response to a command can delegate the function B to application A in order to reduce the need for duplicate codes stored on the IC card. For example, if a PIN needs to be checked before an application is executed, an application stored

ANNEX A TO THE DESCRIPTION

on the card can delegate the "retrieve PIN function" to a PIN application which returns a stored universal PIN for the card.

Preferably, a new session begins whenever the IFD, e.g. a terminal, successfully selects an application, even if the application has been previously selected during the transaction. For example, if a card user goes to a terminal and transfers twenty dollars of electronic cash using a purse application, charges thirty dollars using a credit/debit application and then transfers ten dollars using the purse application again, three separate sessions will have occurred even though only two applications were used during the entire transaction. Each time an application delegates a task or function to another application, the delegated application treats the delegate function as if the IFD devices had selected the application to perform the task or function. However, performing a delegation function as described below has a different effect on session data.

The following examples will help explain when the session data is initialized (i.e., erased) versus when it is saved to be used in further operations. If application A is selected by an IFD device, and receives commands X, Y and Z from the terminal, application A may delegate all three commands to application B. For example, delegations may occur in response to delegation commands in the program code. Both applications A and B will have their session and stack data in their respective Dynamic segments initialized (set to zero) when they receive command X, but the stack will not be initialized when they receive the subsequent commands Y and Z.

In a second example, application A is selected, and receives

ANNEX A TO THE DESCRIPTION

commands X, Y and Z from the terminal. Application A processes X itself, but delegates Y and Z to application B. Application A will have its session and stack data initialized when it receives X, but not when it receives the subsequent commands Y and Z. Application B will have its session and stack data initialized when it receives Y, but not Z.

One example of a use of session data is to support the use of a session Personal Identification Number (PIN). The application could reserve one byte of session data to support the PIN-receiving flag. On receiving the PIN check command, the selected delegated application could update the flag as follows: if the PIN command is received and the inputted PIN is equal to the stored pin, then it will set the session data DB[0] to 1. If not, the application will check if the PIN flag is already set by checking the value in DB[0]. In either of the above cases, the application will process the rest of the commands in the session because the PIN has been verified. If neither of the cases is true, then the application will not process the command because the PIN is not proper. The PIN checking function could be a delegated function from the selected application to a PIN checking application.

The Public segment 105 is used for command and response data being passed between an IFD and an application. During a delegate command, the Public segment contains the data passed between two applications, the delegator (the application initiating the delegation) and the delegated application (the application which performs the delegated function). An application may also use the Public segment as a further temporary working storage space if required. The

ANNEX A TO THE DESCRIPTION

Public data is addressed using offsets stored in register PB 113 as a starting address, to register PT 115 as an ending address. Register PB 113 and Register PT 115 are fixed for the duration of a command-response pair being initiated by the IFD or delegator. Public data can include data inputted into or supplied by a terminal such as a transaction amount, vendor identification data, terminal information, transmission format or other data required or used by an application resident on the IC card. Public data can also include data which is to be transmitted to an IFD device or other application such as an electronic dollar value, card user information transmission format or other data required or used by the terminal or other delegated application.

The multiple application operating system ensures that the data stored in the Public segment remains private to the application until the application exits or delegates. Preferably, the data in the Public segment is then made available to other entities as follows: (1) if the application delegates, the whole of the Public segment becomes available to the delegated application; (2) if the application exits, and is itself delegated by another, the whole of the Public segment becomes available to the delegator; or (3) if the application exits, and is not itself delegated, then a portion of the Public segment containing the I/O response parameters and data are made available to the IFD.

An application may write secret data into the Public memory segment during execution of the application, but the application must make sure it overwrites the secret portion of the Public segment before delegating or exiting. If the application abnormally ends (abends), then the operating system on the IC card preferably overwrites all of the data in the Public segment automatically so that no unwanted entities can have access to the secret data. If the MULTOS carrier device (MCD) is reset, the operating system overwrites data in the Public segment automatically, so that no secret data is revealed. A portion of the Public memory segment is also used as a communications buffer. The I/O protocol data and parameters are preferably stored at the top of the Public memory space. In another preferred embodiment, the top seventeen bytes are reserved for the communications protocol between the IFD device and the IC card application. However, additional or less bytes can also be used depending upon the particular application and operating system being utilized.

The spaces shown between the memory segments in Figure 1 will vary depending upon the specific application and commands being processed. There could be no memory space between the memory segments so that the memory segments are contiguous.

Figure 2 shows an extended illustration of the AAM implemented architecture. Data memory space 201 includes the three segments Static, Public and Dynamic as previously described. Code memory space 203 contains the program instructions for an application stored on the IC card. The application instructions are preferably stored in an executable form which can be interpreted by the resident operating system but can also be stored in machine executable form. Instruction 205 is stored at one location in the code memory space 203. Additional instructions are stored in other locations of memory space 203. Two additional registers 207 and 209 are used in the AAM architecture. A code pointer (CP) register 207

ANNEX A TO THE DESCRIPTION

indicates the particular code instruction to be next executed. In the figure, the register indicates, e.g., through an offset or pointer means, that instruction 205 is the next to be executed. Condition Control Register 209 contains eight bits, four of which are for use by the individual application and four of which are set or cleared depending upon the results of the execution of an instruction. These condition codes can be used by conditional instructions such as Branch, Call or Jump. The condition codes can include a carry bit, an overflow bit, a negative bit and a zero bit.

All address and control registers are set to defined values prior to executing the selected or delegated application. The values are set either when the application is first loaded onto the card and the size of the code and non-volatile data can be ascertained or at the moment when the application passes control to the application. When the application is loaded, SB is set to zero and ST is equal to the number of bytes in the application's Static database. The other address registers are initialized when the application is given control. CP 207 is set to zero and all eight bits in CCR 209 are cleared at the start of executing the application.

A communications interface mechanism is present between the IFD and an application which includes the use of the Public data segment as a communications buffer for command-response parameters. A command-response parameter means an application is given a command to perform and returns a response to the entity issuing the command. Applications interact with an IFD by receiving commands, processing them and returning responses across the IFD-Application Interface. When an application has completed executing a command,

ANNEX A TO THE DESCRIPTION

the application will place the response into the Public segment starting at PB[0] which can be read by the IFD device and will set the proper interface parameters in the reserved Public space relative to PT[0].

While an application can be called directly from an IFD and return a response directly to an IFD, it can also delegate a request to another application where appropriate. The subsequently-called application will then process the request on behalf of the first application. The delegation can be directly in response to a received command in which the delegator acts as a controller for delegating commands or subcommands to other appropriate applications. Alternatively, the delegated command can be embedded in an application's code which delegates control of the processor when the first application needs to interact with another application during its execution, such as updating frequent flyer miles or verifying a PIN.

Figure 3 shows a flow chart of the steps which are performed when a delegate request is executed. Step 301 sets the parameter named delegator_application_id (delegator ID) to be equal to the selected_file.application_id (selected ID). The selected ID indicates the current application which is selected and which is currently being executed. The delegator ID indicates the application which delegates a function to another delegated application stored on the IC card. Step 303 then pushes (stores) the delegator ID onto the top of the delegate_id_stack (delegate stack). The data referenced in the Dynamic portion of allocated memory is saved so that the current application can complete its execution after the delegated function is complete. Data which is to be

ANNEX A TO THE DESCRIPTION

shared with the delegated application is referenced in the Public portion of allocated memory. The delegate stack is preferably stored outside of an application's AAM memory space and keeps track of which applications have delegated functions. Each application is suspended when it delegates a function so the delegate stack can act in a Last-In-First-Out (LIFO) manner so that if a number of applications are suspended due to delegation requests, the proper application is started in the right order. The delegate stack thus keeps track of which application was the last delegator when multiple layered delegation functions are performed. The delegate stack preferably operates in a LIFO manner although different stack schemes could be used as appropriate.

Step 305 then sets the selected ID to the delegate_request.delegate_application_id (delegate ID) value. This step selects the application which will be called to perform the delegated function or functions. The identities of the delegated application can be specifically called by the delegator application or a particular function can be matched up with an application in a look up table. For example, a PIN match operation may be delegated to different applications depending upon which applications are present on the card. Step 307 then sets the application_command parameter to the value stored in the delegate_request.application_command parameter. This step specifies the command to be delegated to the delegate application. Applications typically have the ability to process many different commands. Alternatively, the entire application could be executed to perform one or more functions. The delegator application can choose which command it is delegating to another application. Step 309 then sends the

ANNEX A TO THE DESCRIPTION

application_command to the AAM operating system for execution by the delegatee application. The delegator application is then suspended (or interrupted). Any data that is required to pass between the applications is transferred via the Public memory space.

Figure 4 is a flow chart of the steps for performing a "return delegation control" command by the delegatee application. This command is executed by the operating system when a delegated application has completed its delegated function. Step 401 gets application_responses from the Public memory space of the delegated AAM. The response data is passed in the Public memory segment of the delegatee AAM. Step 403 then sets the delegate_response.status variable to a success condition. This means that a delegation operation has been successfully completed. Step 405 sets the delegate_ response.application_responses parameter to the application_responses values which were stored in the Public segment of the delegatee application.

Step 407 sets the delegate_response.delegate_application_id parameter to selected_file.application_id (the delegatee application ID). Step 409 pops the top (i.e., reads the last data stored in the stack) delegate_application_id from the delegate_id_stack. This information indicates the identity of the delegator application for the command which was just delegated and completed by the delegated application. Step 411 sets the select_file.application_id value to the delegator_application_id value. This selects the delegator application which was identified from the delegate ID stack as the current application which will resume running. The Dynamic data for the delegator application will be retrieved for the

ANNEX A TO THE DESCRIPTION

delegator application from its stored location so that the application will continue to execute where it left off with all data intact but will also have the response information from the delegated function. In step 413, the delegate_response data is sent to the current application for further processing. The response data is passed through the Public data space which could be the same physical RAM memory location because all applications share the physical volatile memory space.

Figure 5 shows a flow chart of the steps involved for inquiring about a delegator ID when a delegate command is received by a delegated application. The delegated application may need to know the identity of the delegator because it may perform operations differently for different delegator applications. For example, an airline loyalty program may need to know if awarded frequent flyers will be based on actual dollars processed or a lump sum award for some other activity such as performing a bill payment operation. This information could be passed to the delegated application as a variable or could be ascertained using an inquiry. The delegator inquiry operation could be implemented as a primitive as previously described.

Step 501 receives the delegator_id_enq_request from the AAM operating system. The request is used to identify the identity of the delegator. Step 503 checks if the delegate_id_stack is empty. If the stack is empty, then no delegation operations have occurred and no applications have been suspended. Thus step 511 sets the delegator_id_enq_response.status parameter to a failure indicator. Step 513 then sets the value of delegator_is_enq_request.error_cause to a value indicating "no delegator application." There is no delegator application. The

ANNEX A TO THE DESCRIPTION

process then continues with step 509.

If the delegate_id_stack is not empty, than one or more delegations have occurred. In that case, step 505 sets the delegator_id_enq_response.status parameter to a value indicating "success". Step 507 then sets the delegator_id_enq_response.delegator_ application_id parameter to the value stored in delegate_id_stack.delegator_ application_id. This sets the inquiry response to indicate the delegator application ID at the top of the stack. As explained above, the stored data at the top of the stack indicates the last delegator application to call a delegate function. Step 509 then sends the delegator_id_enq_ response back to the AAM operator system which delivers the information to the application or IFD entity requesting the information.

Figure 6 shows an example of a block diagram of an integrated circuit located on an IC card chip which can be used in conjunction with the invention. The integrated circuit chip is located on a chip on the card. The IC chip preferably includes a central processing unit 601, a RAM 603, a EEPROM 605, a ROM 607, a timer 609, control logic 611, I/O ports 613 and security circuitry 615, which are connected together by a conventional data bus 617 or other conventional means.

Control logic 611 in the smart card provides sufficient sequencing and switching to handle read-write access to the card's memory through the input/output ports 612. CPU 601 in conjunction with control logic 611 can perform many different functions including performing calculations, accessing memory locations, modifying memory contents, and managing input/output ports. Some IC

ANNEX A TO THE DESCRIPTION

cards also include a coprocessor for handling complex computations like cryptographic algorithms. Input/output ports 613 are used for communication between the card and an IFD which transfers information to and from the card. Timer 609 (which generates and/or provides a clock pulse) drives the control logic 611, CPU 601 and other components requiring a clock signal through the sequence of steps that accomplish functions including memory access, memory reading and/or writing, processing, and data communication. Security circuitry 615 (which is optional) preferably includes fusible links that connect the input/output lines to internal circuitry as required for testing during manufacture, but which are destroyed upon completion of testing to prevent later access. The Static memory space is preferably mapped to memory locations in EEPROM 605 which is non-volatile. The Dynamic memory space is preferably mapped to RAM 603 which is volatile memory which has quick access. The Public memory space is also preferably mapped to RAM 603 which is volatile memory. The Dynamic data and Public data will be stored in different portions of RAM 603, while RAM is identified as a preferred non-volatile memory and EEPROM is identified as a preferred volatile memory. Other types of memory could also be used with the same characteristics.

Figures 7A, 7B and 7C illustrate an example of a delegation function being performed in order to process multiple applications on an IC card. Figure 7A shows a first application being executed as denoted with a double ringed circle 701. At some point during the execution of the first application, a delegation function 702 is called to delegate an operation to the second application which is indicated

ANNEX A TO THE DESCRIPTION

by circle 703. Also shown in Figure 7A is an empty delegator ID stack 705. Since the stack is empty, there is no data associated with it and it is shown only for illustrative purposes.

The multiple application operating system receives the delegate command and interrupts the execution of the first application 701 and gives control of the integrated circuit to application 703 as shown in Figure 7B. The execution of the second application 703 is illustrated with a double ringed circle. The term "gives control" means that the microprocessor and other circuitry on the card will process the instructions and allocate memory space for the application which is delegated. When the delegate command is processed, the delegator ID 707 is placed on top of the stack 705. The delegator ID stack is operated in a LIFO manner. Also shown in Figure 7B is a third application 709 resident on the card. At some point during the execution of the second application, a delegate function 711 is called to delegate the operation to the third application.

The multiple application operating system receives the delegate command 711 shown in Figure 7B interrupts the execution of the second application 703 and gives control of the integrated circuit to the third application 709 as shown in Figure 7C. When the delegate command is processed, the delegator ID 713 of the second application is pushed onto the delegator ID stack 705. The delegator ID 707 of the first application whose execution is still interrupted is pushed down in the stack consistent with a LIFO stack management. Thus when the third application has finished its execution, the delegator ID at the top of the stack is popped to indicate that execution of the second application

ANNEX A TO THE DESCRIPTION

should be resumed first. The delegator ID 707 from the first application will then be at the top of the stack so that when the second application is finished executing, the first application will resume its execution.

Additional applications can be managed by the delegator ID stack in a similar manner. By interrupting the execution of the applications when a delegate command is processed and keeping track of the order of delegations, the security and integrity of the data for each individual application can be maintained which is important because IC cards will store data for applications which is private to the card user such as account numbers, social security number, address and other personal information.

The foregoing merely illustrates the principles of the invention. It will thus be appreciated that those skilled in the art will be able to devise numerous apparatus, systems and methods which, although not explicitly shown or described herein, embody the principles of the invention and are thus within the spirit and scope of the invention.

ANNEX A TO THE DESCRIPTION

WE CLAIM:

1. An integrated circuit card comprising:

a microprocessor; a volatile memory coupled to said microprocessor; a non-volatile memory coupled to said microprocessor; and a plurality of applications stored in said non-volatile memory, wherein upon execution of each said application, said microprocessor allocates for each said executing application an associated data memory space comprising at least a volatile memory segment for referencing temporary data and a non-volatile memory segment for referencing static data; and further comprising means for delegating the performance of a function from a first executing application to a second executing application.

2. The integrated circuit card of claim 1, wherein said non-volatile memory segment is divided into at least two regions, including a public region and a dynamic region.

3. The integrated circuit card of claim 2, wherein said public region is used to share data between said first and second applications.

4. The integrated circuit card of claim 2, wherein said dynamic region is used to reference temporary data utilized during an application's execution.

ANNEX A TO THE DESCRIPTION

5. The integrated circuit card of claim 1, further comprising at least one register coupled to said microprocessor which is used to determine the starting locations of each of said segments.

6. The integrated circuit card of claim 5, further comprising at least one register coupled to said microprocessor which is used to determine the top locations of each of said segments.

7. The integrated circuit card of claim 6, further comprising at least one register coupled to said microprocessor which is used as a local dynamic pointer.

8. The integrated circuit card system of claim 1, wherein each said application comprise a plurality of program instructions and wherein at least one of said program instructions when executed causes said memory referenced by said volatile memory segment to be accessed.

9. The integrated circuit card of claim 1, wherein said volatile memory segment references RAM and said non-volatile memory segment references EEPROM.

10. A method for processing a plurality of applications stored in a memory of an integrated circuit:

selecting a first application for execution;

ANNEX A TO THE DESCRIPTION

allocating a data space for said first application including at least two memory segments comprising a volatile memory segment for referencing temporary data and a non-volatile memory segment for referencing static data;

executing said first application, interrupting execution of said first application and saving data referenced by said volatile memory segment;

executing a second application;

utilizing said saved data from said volatile memory segment for execution of said first application; and

completing said execution of said first application.

11. The method of claim 10, wherein said first application's identity is stored in a data stack during said delegation step.

12. The method of claim 11, wherein said data stack is accessed following said completion of said second application.

13. The method of claim 12, further including the step of inquiring said first application's identity by accessing said delegator stack.

14. The method of claim 10, wherein said non-volatile memory segment is divided into at least two regions, including a public region and a dynamic region.

ANNEX A TO THE DESCRIPTION

15. The method of claim 14, wherein said public region is used to share data between said first application and said second application.

16. The method of claim 14, wherein data referenced by said dynamic region is utilized during the execution of said first application.

17. The method of claim 10, further including the step of allocating a second data space including at least two memory segments for said second application.

18. The method of claim 17, wherein said second data space's segments comprise a volatile memory segment for referencing temporary data and a non-volatile memory segment for referencing static data.

19. The method of claim 18, wherein said second application's non-volatile segment is divided into at least two regions, including a public region and a dynamic region.

20. The method of claim 19, wherein said second application's public region is used to share data between said first and second applications.

ANNEX A TO THE DESCRIPTION

21. The method of claim 19, wherein said data referenced by second application's dynamic region is utilized during said execution of said second application.

22. The method of claim 10, further including the step of delegating use of said microprocessor from said second application to a third application stored on said IC card.

23. The method of claim 22, wherein a third data space for said third application is allocated which includes a volatile memory segment for referencing temporary data and non-volatile memory segment for referencing static data, wherein said third application's volatile segment includes a public and dynamic portion.

24 An apparatus for processing a plurality of applications stored in a memory of a single integrated circuit card comprising:

means for allocating a data space comprising at least a non-volatile memory segment for referencing static data and a volatile memory segment for referencing temporary data; means for executing a first application; means for interrupting execution of said first application, means for saving data from at least a portion of said volatile memory segment; and means for executing a second application; means for retrieving said saved data; and means for completing said execution of said first application.

ANNEX A TO THE DESCRIPTION

25. The apparatus of claim 24, further including means for storing said first application's identity on a data stack.

26. The apparatus of claim 25, further including means for inquiring of said first application's identity.

27. The apparatus of claim 24, wherein said first application's non-volatile memory segment is divided into at least two regions, including a public region and a dynamic region.

28. The apparatus of claim 27, wherein said public region references random access memory.

29. The apparatus of claim 27, wherein said dynamic region references random access memory.

30. The apparatus of claim 24, further including means for allocating a second data space including at least two segments for said second application.

31. The apparatus of claim 30, wherein said second data space includes a volatile memory segment for referencing temporary data and a non-volatile memory segment for referencing static data.

ANNEX A TO THE DESCRIPTION

32. The apparatus of claim 31, wherein said second data space's non-volatile segment is divided into at least two regions, including a public region and a dynamic region.

33. The apparatus of claim 32, wherein said public region references random access memory.

34. The apparatus of claim 32, wherein said dynamic region references random access memory.

35. The apparatus of claim 24, further including means for delegating operation of said IC card from said second application to a third application stored on said IC card.

36. The apparatus of claim 35, wherein a third data space for said third application is allocated which includes a volatile memory segment for referencing temporary data and non-volatile memory segment for referencing temporary data, wherein said third application's volatile memory segment includes a public and dynamic portion.

37. A system for processing a plurality of applications stored on an IC card comprising:

a non-volatile memory coupled to a databus;

ANNEX A TO THE DESCRIPTION

a volatile memory coupled to said databus;

a first and second application program stored in said non-volatile memory, wherein each application has an associated identifier;

a data stack accessible by said databus for storing said applications' identifier if said application is interrupted during its execution;

processor means for executing instructions from said application programs wherein said processor means allocates a data memory space for said application which is being executed and said data memory space is mapped to at least one address in said non-volatile memory and at least one address in said volatile memory; and

wherein said processor means interrupts said first application at least once during its execution to execute said second application.

38. The system of claim 37, wherein data memory space comprises at least a volatile memory segment for referencing temporary data stored in said volatile memory and a non-volatile memory segment for referencing static data stored in said non-volatile memory.

39. The system of claim 37, further including means for storing said first application's identity on a data stack.

40. The system of claim 39, further including means for inquiring of said first application's identity.

ANNEX A TO THE DESCRIPTION

41. The system of claim 38, wherein said first application's non-volatile memory segment is divided into at least two regions, including a public region and a dynamic region.

42. The system of claim 41, wherein said public region references random access memory.

43. The system of claim 41, wherein said dynamic region references random access memory.

44. The system of claim 37, further including means for allocating a second data space including at least two segments for said second application.

45. The system of claim 44, wherein said second data space comprises at least a volatile memory segment for referencing temporary data and a non-volatile memory segment for referencing static data.

46. The system of claim 45, wherein said second data space's non-volatile segment is divided into at least two regions, including a public region and a dynamic region.

47. The system of claim 46, wherein said public region references random access memory.

ANNEX A TO THE DESCRIPTION

48. The system of claim 46, wherein said dynamic region references random access memory.

49. The system of claim 37, further including means for delegating use of said processor means from said second application to a third application stored on said IC card.

50. The system of claim 49, wherein a third data space for said third application is allocated which includes a volatile memory segment for referencing temporary data and non-volatile memory segment for referencing temporary data, wherein said third application's volatile memory segment includes a public and dynamic portion.

51. An integrated circuit card comprising:

a plurality of applications and a microprocessor for controlling execution of said applications wherein execution of at least one first application is interrupted and execution is transferred to another second application, further comprising means for sharing data by said first and second applications and means for resuming execution of said first application at the appropriate location at least after completion of execution of said second application.

ANNEX A TO THE DESCRIPTION

52. The integrated circuit card of claim 51, further comprising means for allocating a data memory space comprises at least a volatile memory segment for referencing temporary data stored in said volatile memory and a non-volatile memory segment for referencing static data stored in said non-volatile memory.

53. The integrated circuit card of claim 51, further including means for storing said first application's identity on a data stack.

54. The integrated circuit card of claim 53 further including means for inquiring of said first application's identity.

55. The integrated circuit card of claim 52, wherein said first application's non-volatile memory segment is divided into at least two regions, including a public region and a dynamic region.

56. The integrated circuit card of claim 55, wherein said public region references random access memory.

57. The integrated circuit card of claim 55, wherein said dynamic region references random access memory.

ANNEX A TO THE DESCRIPTION

58. The integrated circuit card of claim 52, further including means for allocating a second data space including at least two segments for said second application.

59. The integrated circuit card of claim 58, wherein said second data space comprises at least a volatile memory segment for referencing temporary data and a non-volatile memory segment for referencing static data.

60. The integrated circuit card of claim 58, wherein said second data space's non-volatile segment is divided into at least two regions, including a public region and a dynamic region.

61. The integrated circuit card of claim 58, wherein said public region references random access memory.

62. The integrated circuit card of claim 60, wherein said dynamic region references random access memory.

63. The integrated circuit card of claim 51, further including means for delegating use of said processor means from said second application to a third application stored on said IC card.

ANNEX A TO THE DESCRIPTION

ABSTRACT OF THE DISCLOSURE

A multi-application IC card which processes two or more applications using an Application Abstract Machine architecture. The AAM architecture only allows one application to be executed at a time and allows for shared processing by performing a delegation function to a second application. A data space for each application is allocated when the application is selected to be executed. The data space includes a volatile and non-volatile region. The delegation function temporarily interrupts the execution of the first application, saves the temporary data of the first application, shares any data needed with the second application and the second application is executed until the delegated task is competed. The first application then retrieves the saved data and completes its execution. A delegator stack is used to keep track of the delegator's identity when multiple delegations occur. The AAM model allows for a high level of security while transferring data between applications.

CLAIMS

I CLAIM:

1. An integrated circuit card having an associated operating mode, comprising:

a microprocessor;

a memory coupled to said microprocessor;

data stored in said memory representative of said operating mode;

an operating system stored in said memory for processing selected information in a first IC card format;

a shell application stored in said memory for processing said selected information in a second IC card format; and

means responsive to said operating mode for routing said selected information to either said operating system or said shell application.

2. The integrated circuit card of claim 1, wherein said second IC card format is different than said first IC card format.

3. The integrated circuit card of claim 1 or claim 2, wherein said selected information is a command.

4. The integrated circuit card of claim 3, wherein said command is a file access command.

5. The method of any preceding claim, wherein said selected information is associated with a file structure format.

6. The integrated circuit card of any preceding claim, further comprising:

a non-shell application stored in said memory;

means for receiving a request by said operating system from said non-shell application for delegating control to a delegated application;

means for determining whether said operating mode of said IC card is a predetermined operating mode;

means for determining whether said delegated application corresponds to said shell application; and

means for failing the request for delegating control if the operating mode of said IC card corresponds to said predetermined operating mode and said delegated application corresponds to said shell application.

7. A method of loading an application onto an IC card, wherein said application has an associated file mode type and said IC card has an associated operating mode, comprising the steps of:

determining whether the file mode type of said application is a predetermined file mode type; and

changing the operating mode of said IC card if said file mode type corresponds to said predetermined file mode type.

8. The method of claim 7, further comprising the step of determining whether any other applications have already been loaded onto the IC card before the step of changing the operating mode.

9. The method of claim 7 or claim 8, further comprising loading said application onto the IC card if the file mode type of said application corresponds to the predetermined file mode type and no other applications have already been loaded onto the IC card.

10. The method of claim 8, wherein the changing step comprises changing the operating mode of said IC card if said file mode type corresponds to said predetermined file mode type and no other applications have already been loaded onto the IC card.

11. A method of routing a command by an operating system of an IC card, wherein said IC card has an associated operating mode, comprising the steps of:

determining whether the operating mode of said IC card is a predetermined operating mode; and

routing the command directly to an application if the operating mode of said IC card corresponds to the predetermined operating mode.

12. The method of claim 11, further comprising the steps of:

if the operating mode of said IC card does not correspond to the predetermined operating mode, determining whether said command is a select file command supported by said operating system; and

routing said command to an operating system routine responsible for said select file command if said command is a select file command supported by said operating system.

13. The method of claim 11 or claim 12, wherein the IC card further comprises a currently selected file having an associated file type, the method further comprising the steps of:

if the operating mode of said IC card does not correspond to the predetermined operating mode, determining whether the file type of said currently selected file is supported by said operating system; and

routing said command to an operating system routine responsible for said file type if the file type of said currently selected file is supported by said operating system.

14. The method of claim 13, if the file type of said currently selected file is not supported by said operating system, further comprising the step of routing said command to an application.

15. A method of delegating control between applications by an operating system of an IC card, wherein said IC card is for use with a defined IC card format and has an associated operating mode, comprising the steps of:

storing a shell application in said IC card for communicating with said operating system and for processing information in a format compliant with said defined IC card format;

receiving a request by said operating system from a first application for delegating control to a second application;

determining whether the operating mode of said IC card is a predetermined operating mode;

determining whether said second application corresponds to said shell application; and

failing the request for delegating control if the operating mode of said IC card corresponds to said predetermined operating mode and said second application corresponds to said shell application.

16. A method of initiating communication between an IC card and a terminal, wherein said IC card comprises a microprocessor and a memory, said memory having stored therein an operating system, a shell application, and data

representative of an operating mode of said IC card, said operating mode representing whether selected information is to be routed to said operating system or said shell application, said method comprising the steps of:

receiving a reset signal by said IC card from said terminal; and

returning an answer-to-reset from said IC card to said terminal based on said operating mode of said IC card.

17. The method of claim 16, wherein a plurality of answer-to-reset files are stored in said memory of said IC card, and said step of returning an answer-to-reset comprises selecting one of said answer-to-reset files based on said operating mode.

18. The method of claim 16 or claim 17, wherein said selected information is a command.

19. The method of claim 18, wherein said command is a file access command.

20. The method of claim 16, wherein said selected information is associated with a file structure format.

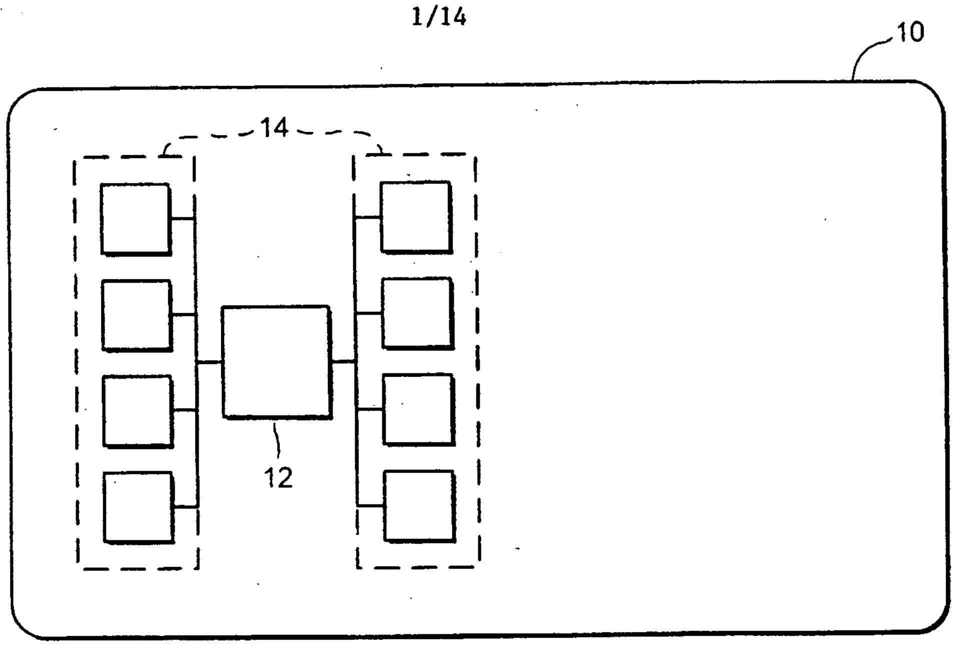


FIG. 1

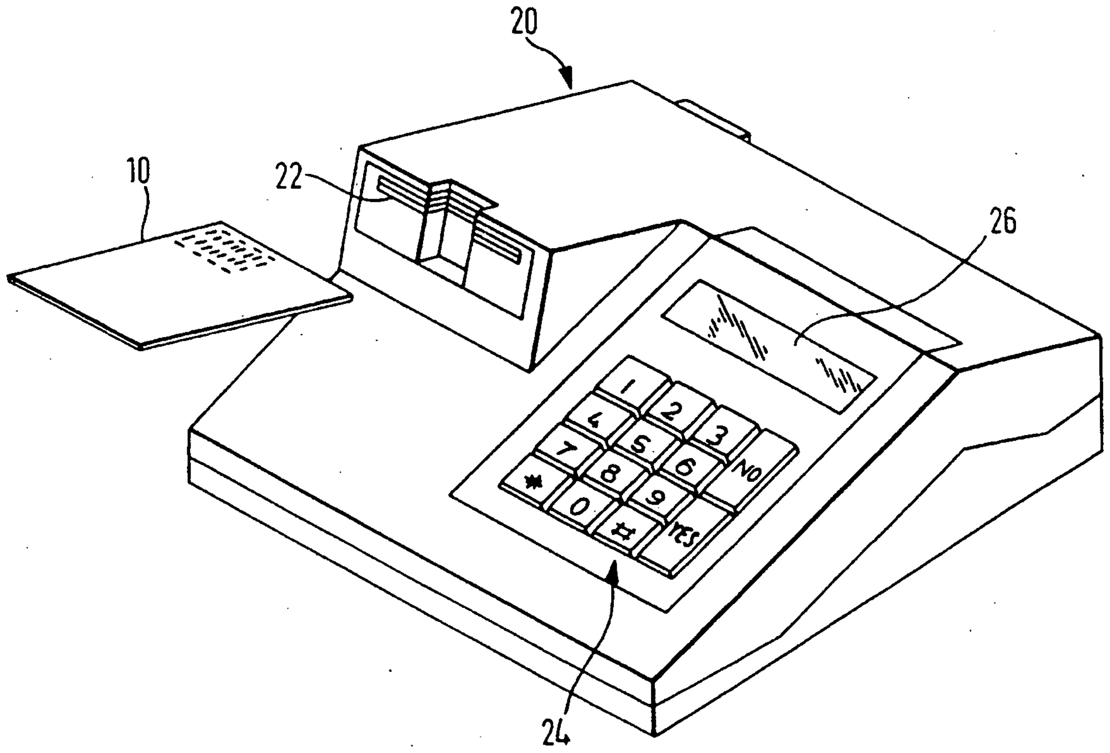


FIG. 2

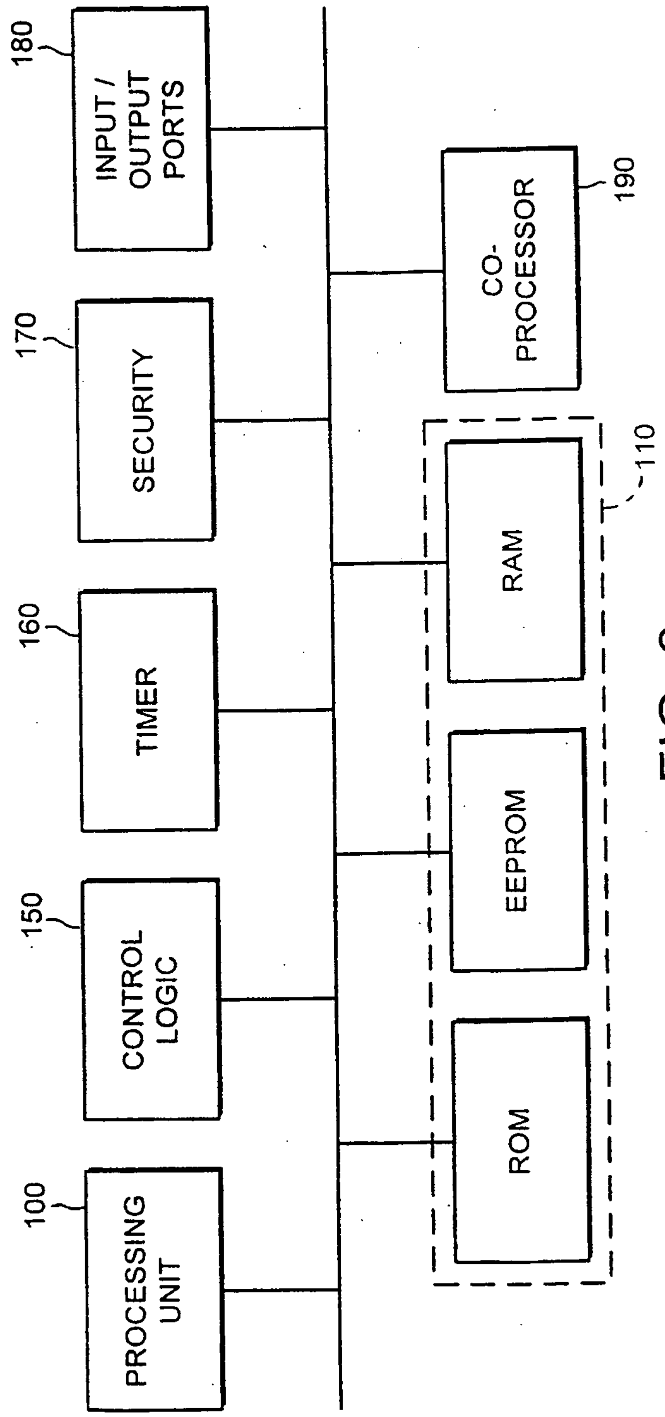

180
INPUT / OUTPUT PORTS
170
SECURITY
160
TIMER
150
CONTROL LOGIC
100
PROCESSING UNIT
190
CO-PROCESSOR
110
RAM
EEPROM
ROM

FIG. 3

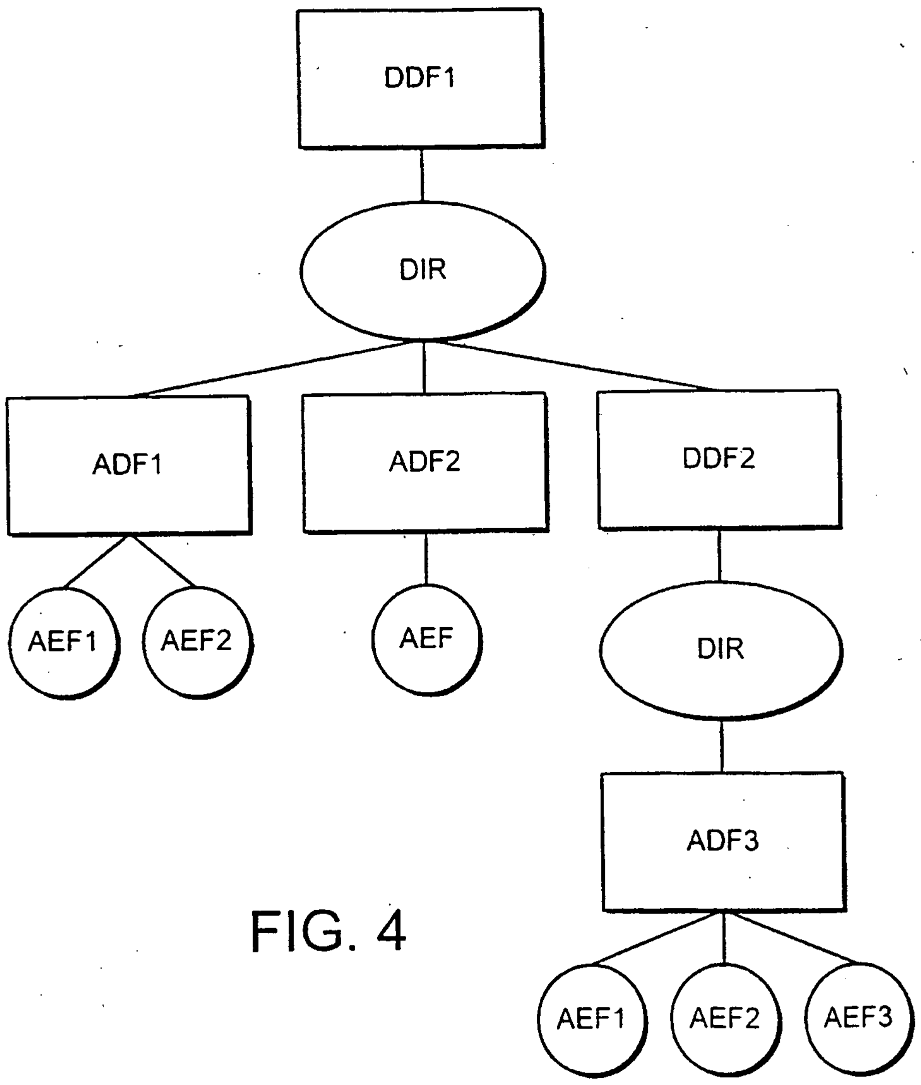


FIG. 4

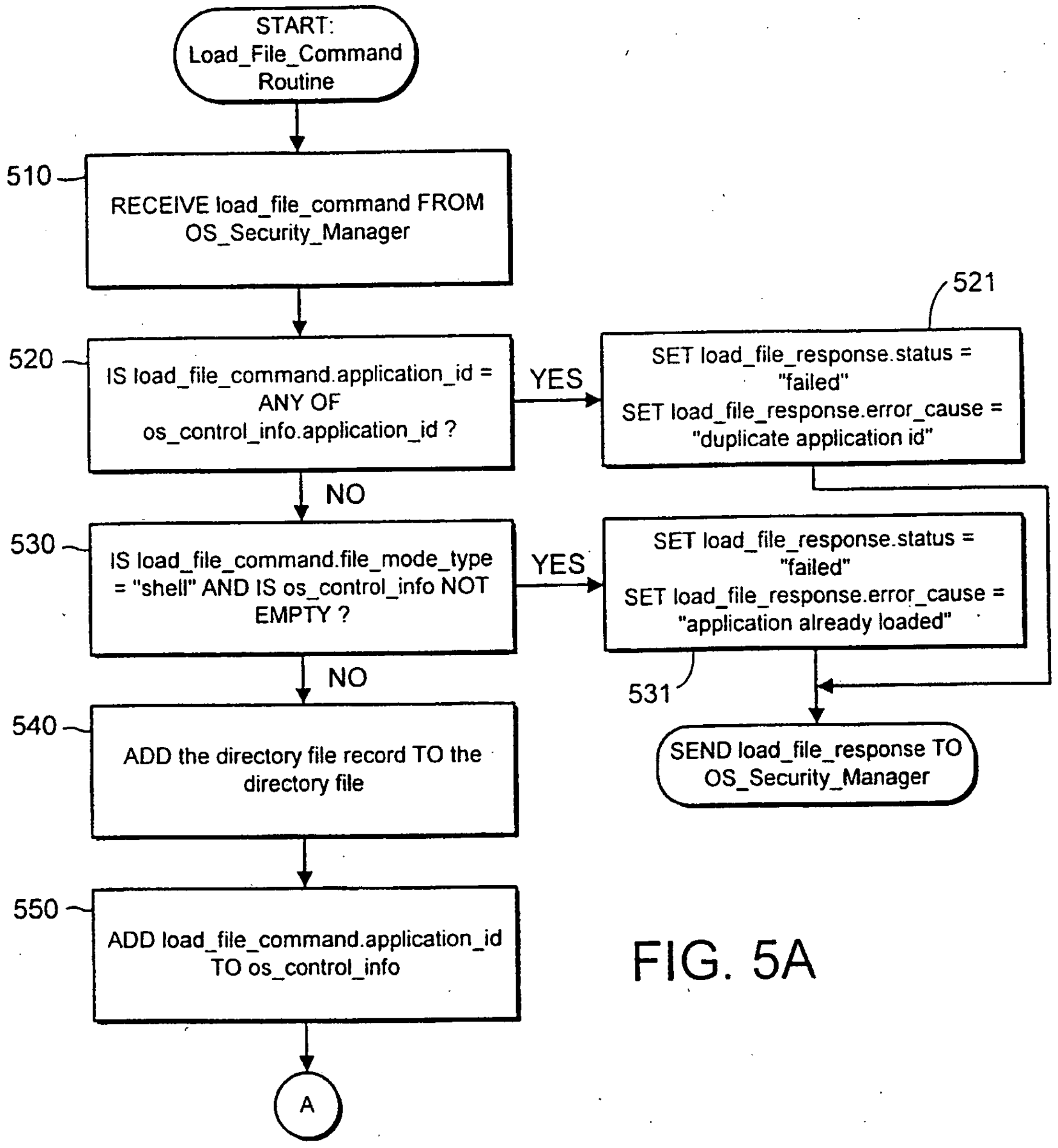

START: Load_File_Command Routine
510
RECEIVE load_file_command FROM OS_Security_Manager
520
IS load_file_command.application_id = ANY OF os_control_info.application_id ?
YES
521
SET load_file_response.status = "failed"
SET load_file_response.error_cause = "duplicate application id"
NO
530
IS load_file_command.file_mode_type = "shell" AND IS os_control_info NOT EMPTY ?
YES
SET load_file_response.status = "failed"
SET load_file_response.error_cause = "application already loaded"
531
NO
540
ADD the directory file record TO the directory file
SEND load_file_response TO OS_Security_Manager
550
ADD load_file_command.application_id TO os_control_info
A

FIG. 5A

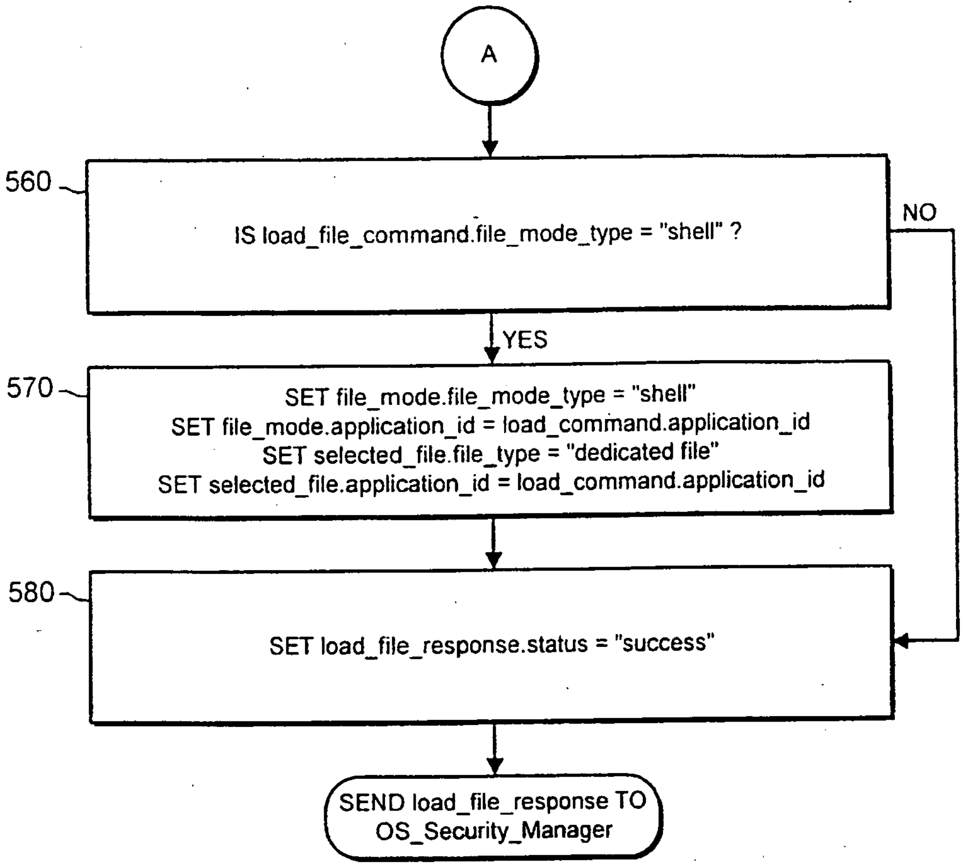

A
560
IS load_file_command.file_mode_type = "shell" ?
NO
YES
570
SET file_mode.file_mode_type = "shell"
SET file_mode.application_id = load_command.application_id
SET selected_file.file_type = "dedicated file"
SET selected_file.application_id = load_command.application_id
580
SET load_file_response.status = "success"
SEND load_file_response TO OS_Security_Manager

FIG. 5B

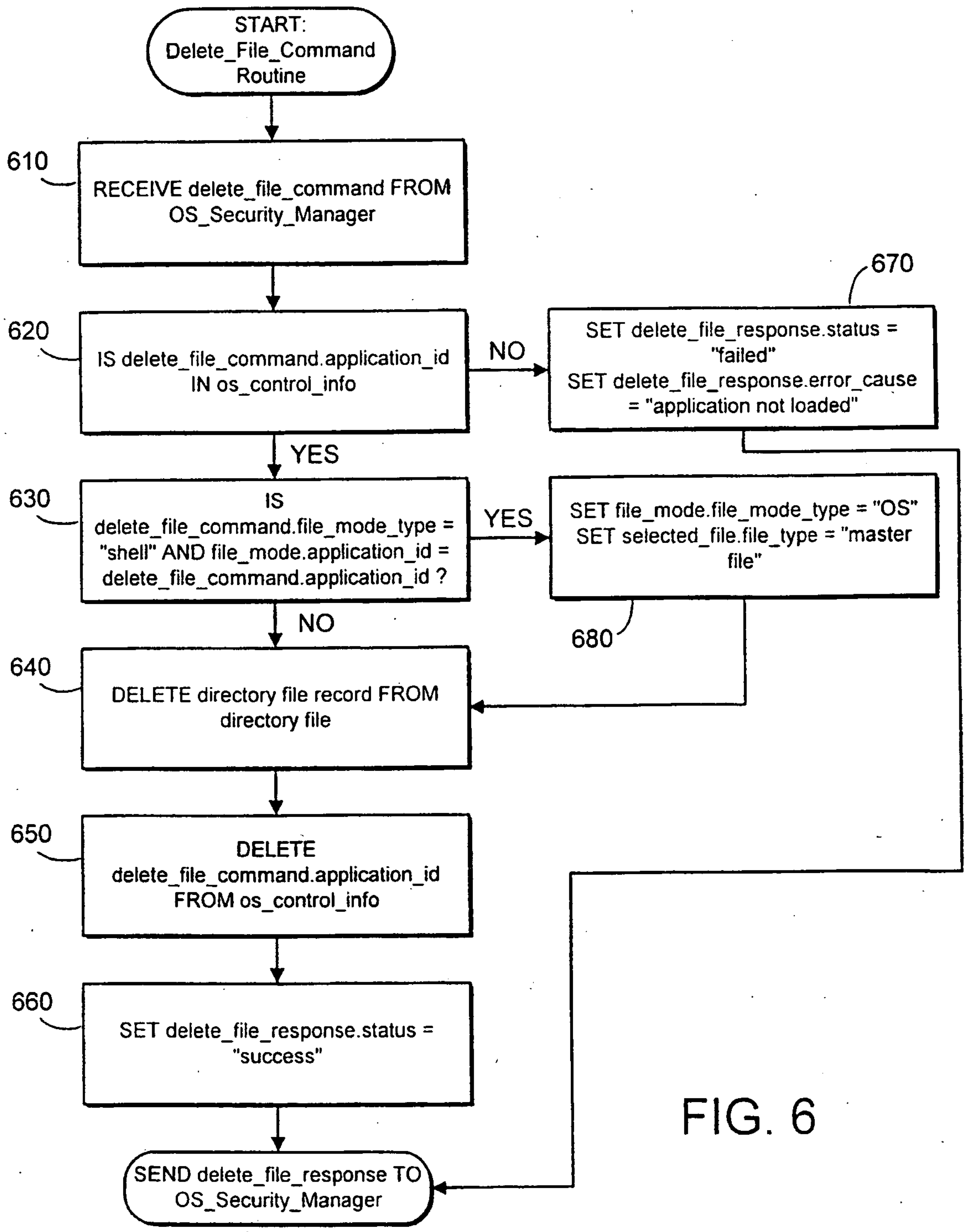

START: Delete_File_Command Routine
610
RECEIVE delete_file_command FROM OS_Security_Manager
620
IS delete_file_command.application_id IN os_control_info
NO
670
SET delete_file_response.status = "failed" SET delete_file_response.error_cause = "application not loaded"
YES
630
IS delete_file_command.file_mode_type = "shell" AND file_mode.application_id = delete_file_command.application_id ?
YES
SET file_mode.file_mode_type = "OS" SET selected_file.file_type = "master file"
680
NO
640
DELETE directory file record FROM directory file
650
DELETE delete_file_command.application_id FROM os_control_info
660
SET delete_file_response.status = "success"
SEND delete_file_response TO OS_Security_Manager

FIG. 6

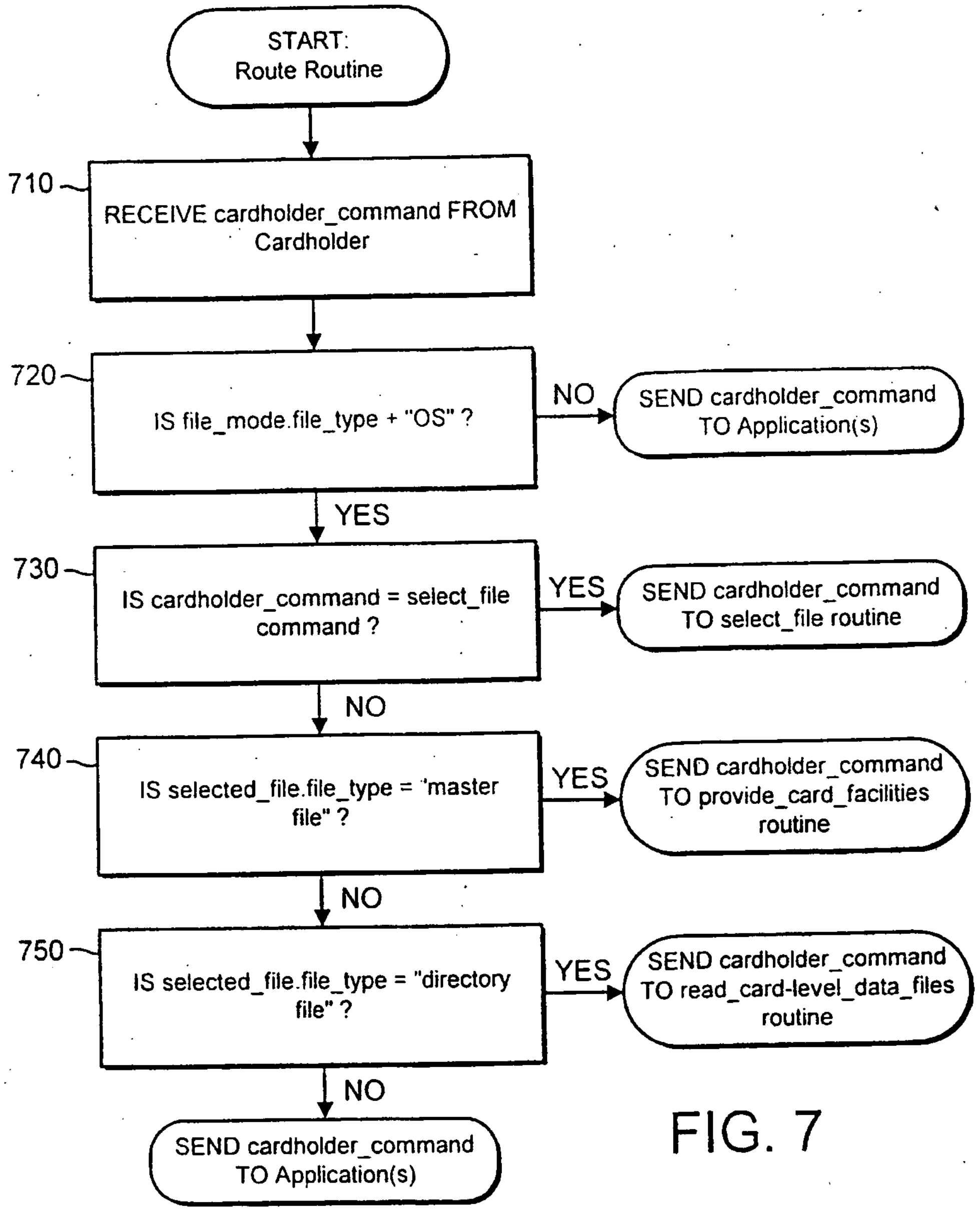

START:
Route Routine
710
RECEIVE cardholder_command FROM Cardholder
720
IS file_mode.file_type + "OS" ?
NO
SEND cardholder_command TO Application(s)
YES
730
IS cardholder_command = select_file command ?
YES
SEND cardholder_command TO select_file routine
NO
740
IS selected_file.file_type = 'master file" ?
YES
SEND cardholder_command TO provide_card_facilities routine
NO
750
IS selected_file.file_type = "directory file" ?
YES
SEND cardholder_command TO read_card-level_data_files routine
NO
SEND cardholder_command TO Application(s)

FIG. 7

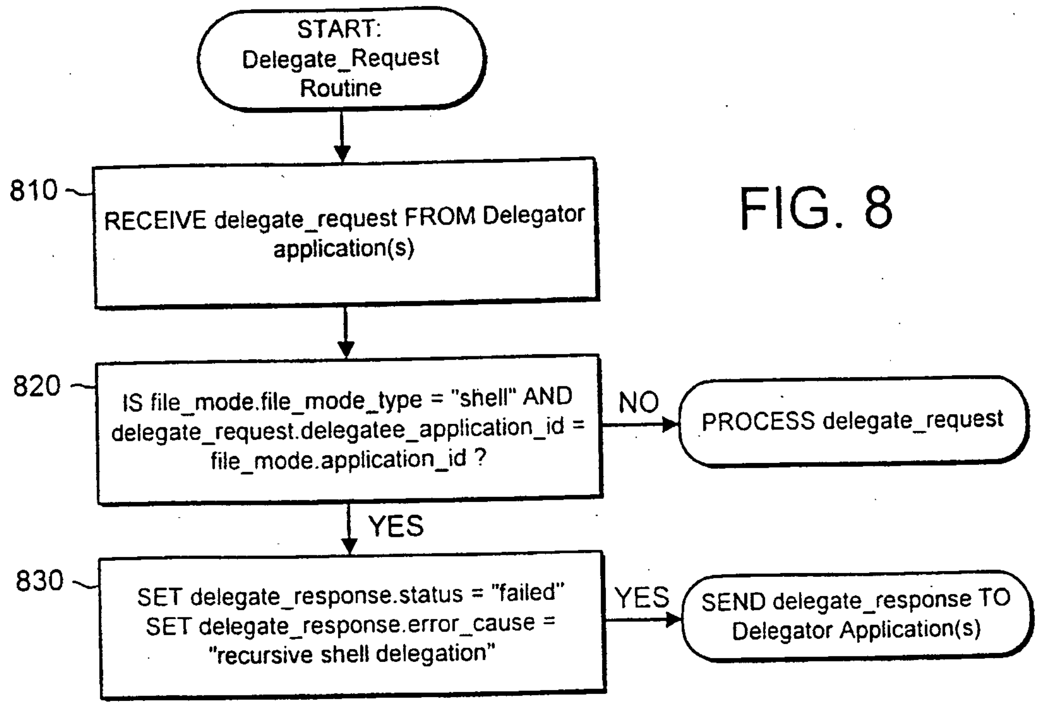

START: Delegate_Request Routine
810
RECEIVE delegate_request FROM Delegator application(s)
820
IS file_mode.file_mode_type = "shell" AND delegate_request.delegatee_application_id = file_mode.application_id ?
NO
PROCESS delegate_request
YES
830
SET delegate_response.status = "failed" SET delegate_response.error_cause = "recursive shell delegation"
YES
SEND delegate_response TO Delegator Application(s)

FIG. 8

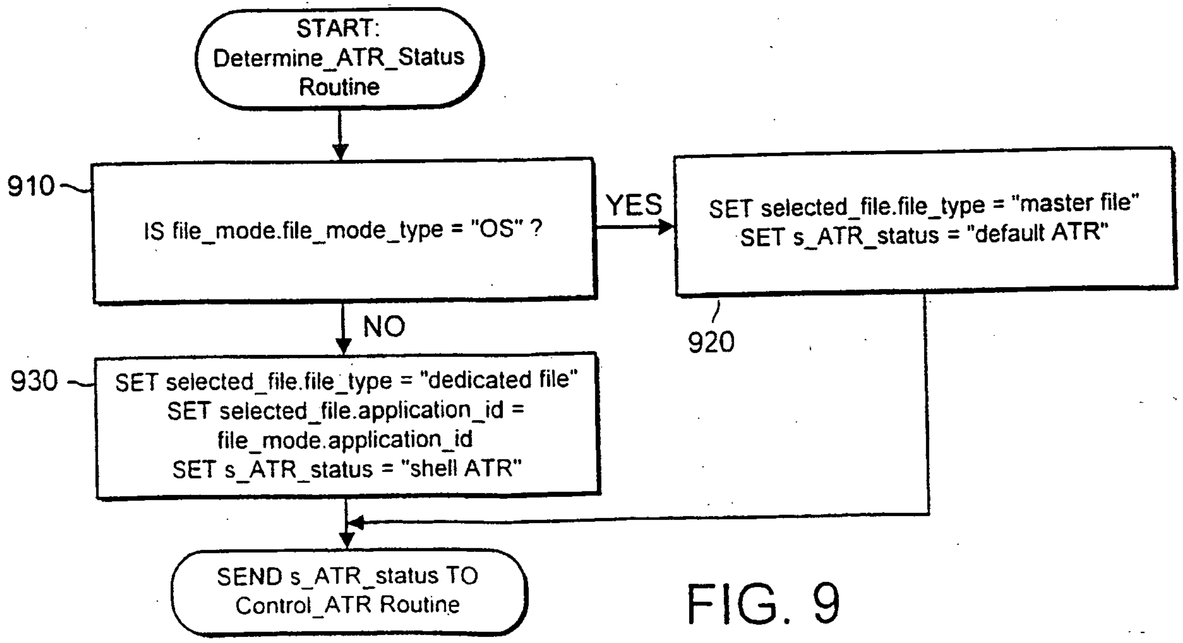

START: Determine_ATR_Status Routine
910
IS file_mode.file_mode_type = "OS" ?
YES
SET selected_file.file_type = "master file" SET s_ATR_status = "default ATR"
920
NO
930
SET selected_file.file_type = "dedicated file" SET selected_file.application_id = file_mode.application_id SET s_ATR_status = "shell ATR"
SEND s_ATR_status TO Control_ATR Routine

FIG. 9

ANNEX A TO THE DRAWINGS

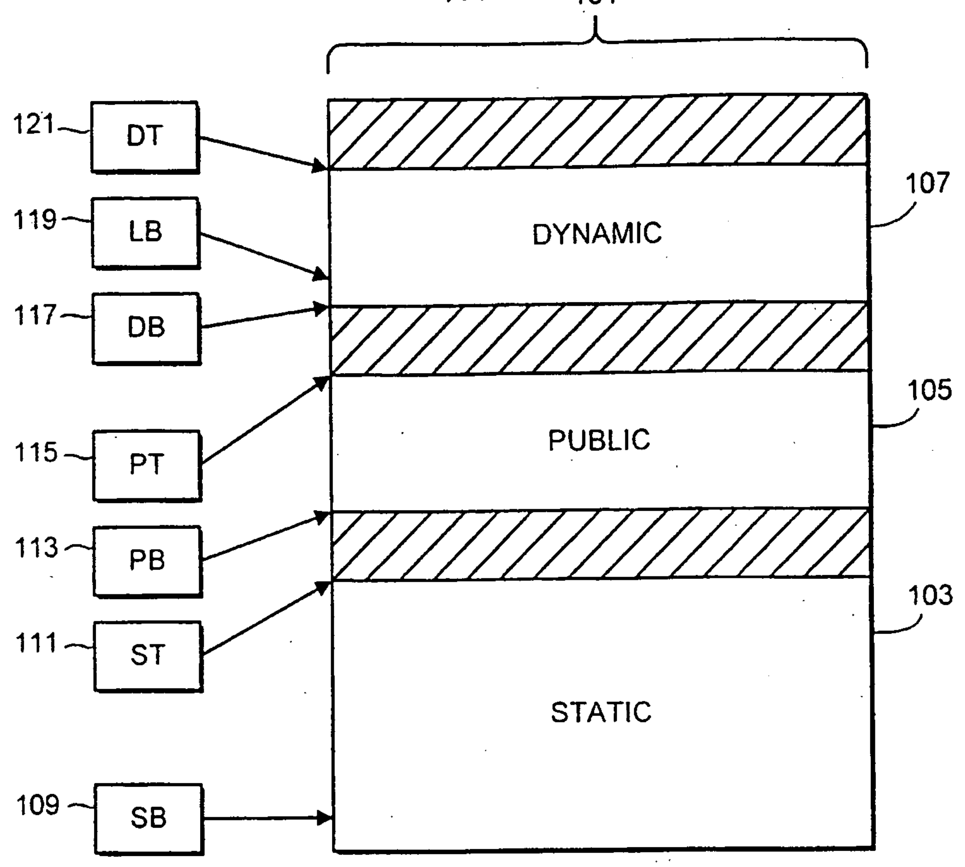


FIG. 1

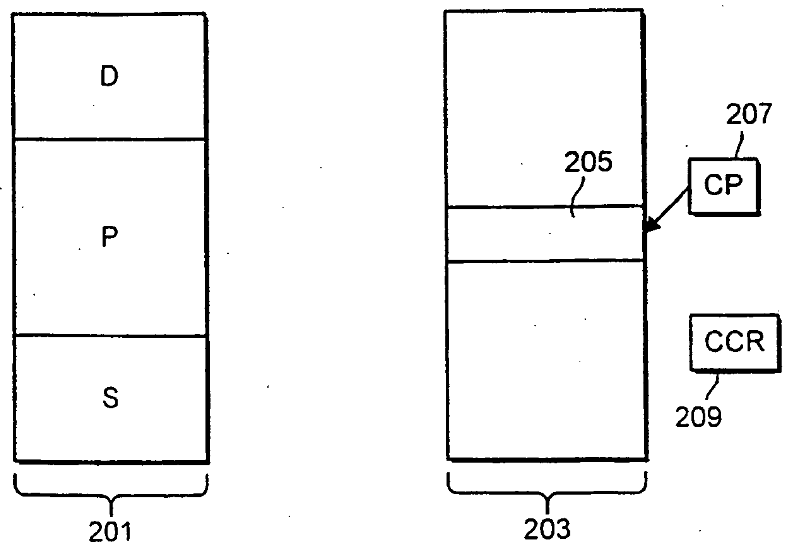


FIG. 2

ANNEX A TO THE DRAWINGS

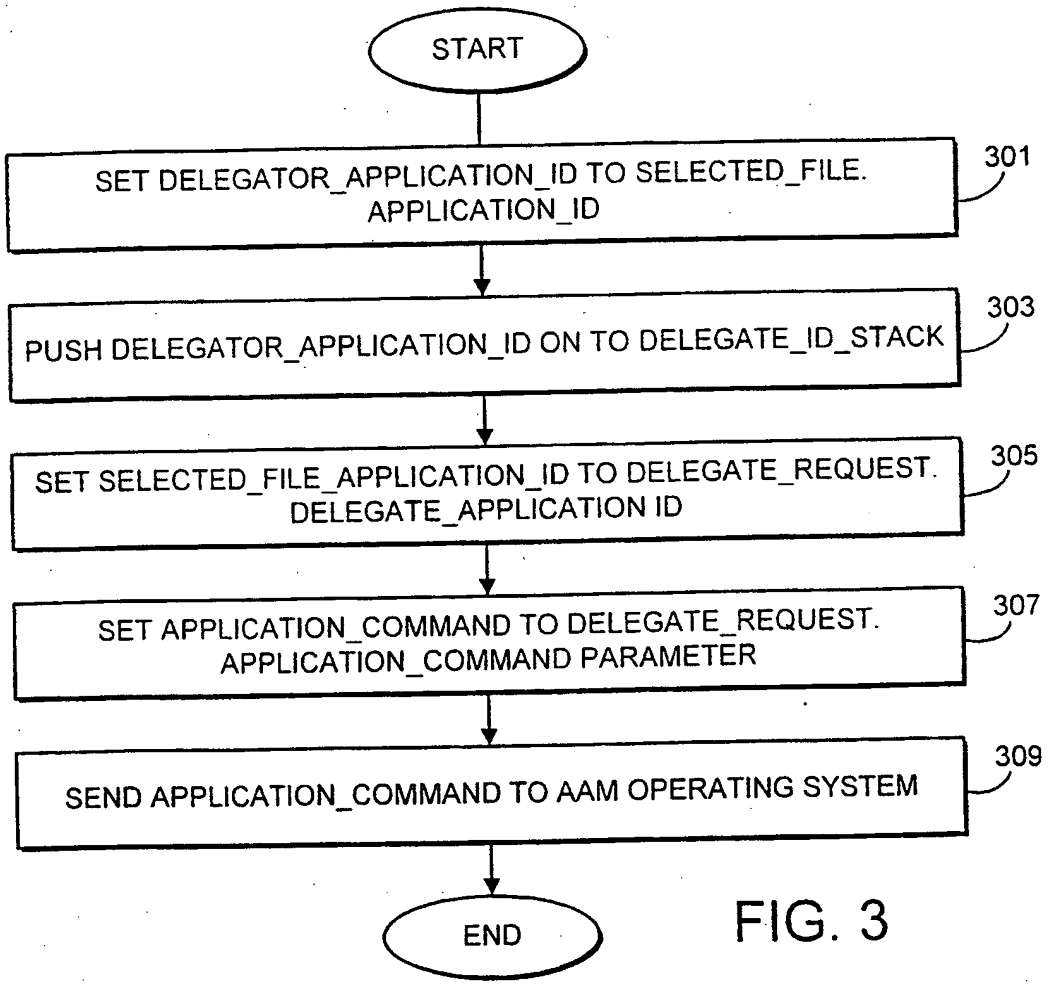
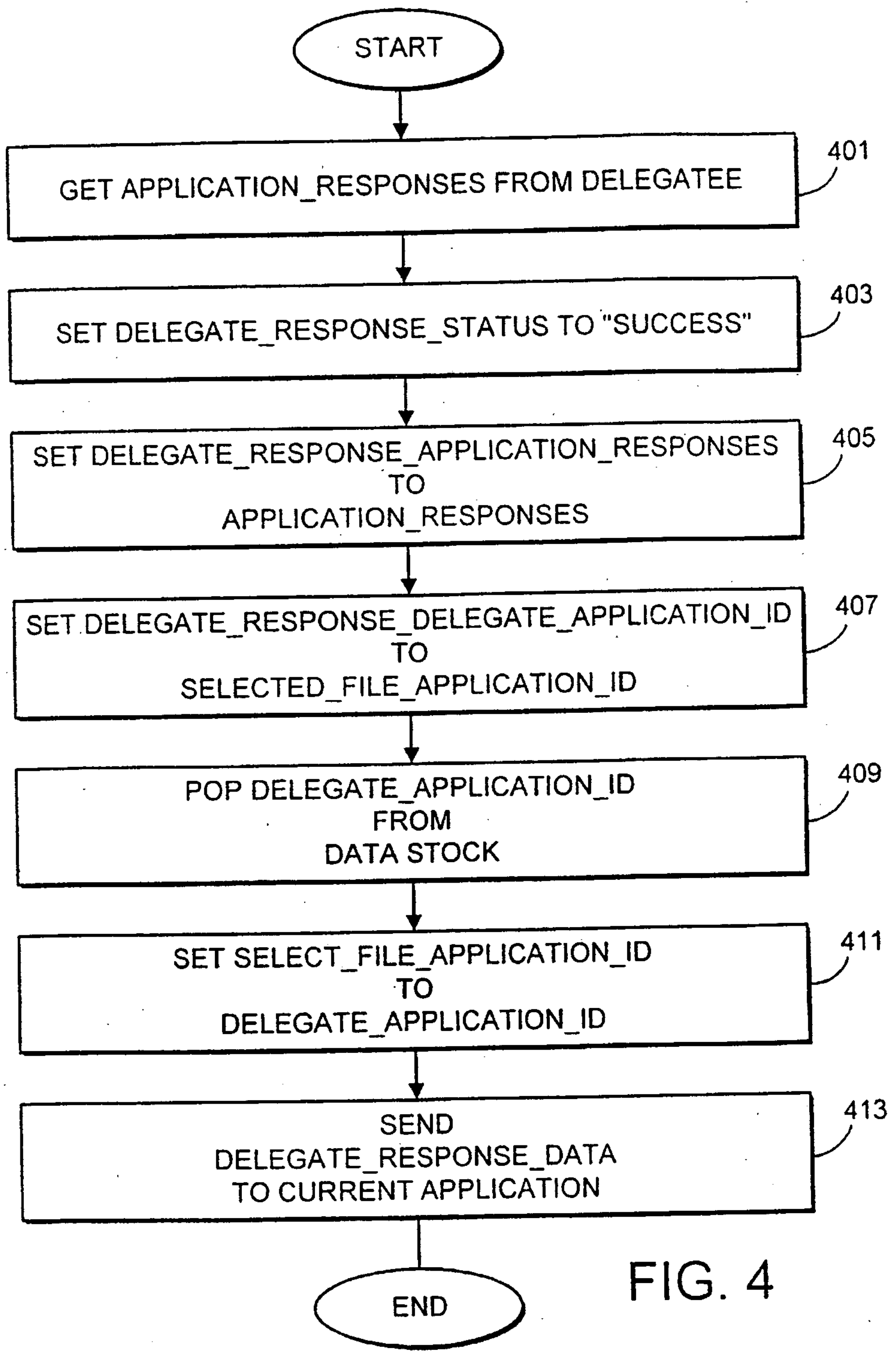

START
SET DELEGATOR_APPLICATION_ID TO SELECTED_FILE. APPLICATION_ID
301
PUSH DELEGATOR_APPLICATION_ID ON TO DELEGATE_ID_STACK
303
SET SELECTED_FILE_APPLICATION_ID TO DELEGATE_REQUEST. DELEGATE_APPLICATION ID
305
SET APPLICATION_COMMAND TO DELEGATE_REQUEST. APPLICATION_COMMAND PARAMETER
307
SEND APPLICATION_COMMAND TO AAM OPERATING SYSTEM
309
END

FIG. 3

ANNEX A TO THE DRAWINGS



START
GET APPLICATION_RESPONSES FROM DELEGATEE
401
SET DELEGATE_RESPONSE_STATUS TO "SUCCESS"
403
SET DELEGATE_RESPONSE_APPLICATION_RESPONSES TO APPLICATION_RESPONSES
405
SET DELEGATE_RESPONSE_DELEGATE_APPLICATION_ID TO SELECTED_FILE_APPLICATION_ID
407
POP DELEGATE_APPLICATION_ID FROM DATA STOCK
409
SET SELECT_FILE_APPLICATION_ID TO DELEGATE_APPLICATION_ID
411
SEND DELEGATE_RESPONSE_DATA TO CURRENT APPLICATION
413
END

FIG. 4

ANNEX A TO THE DRAWINGS

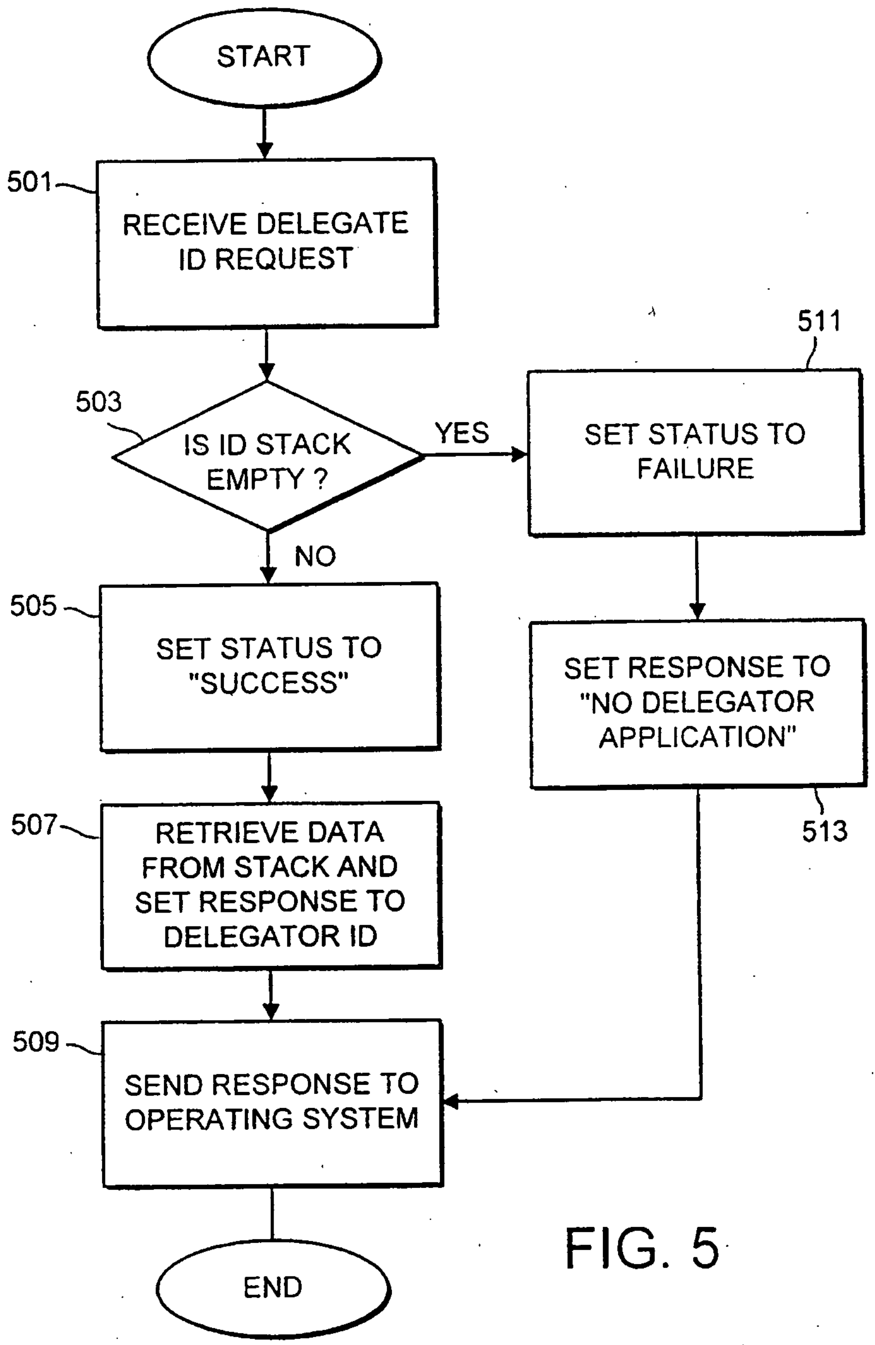

START
501
RECEIVE DELEGATE ID REQUEST
503
IS ID STACK EMPTY ?
YES
NO
511
SET STATUS TO FAILURE
505
SET STATUS TO "SUCCESS"
SET RESPONSE TO "NO DELEGATOR APPLICATION"
513
507
RETRIEVE DATA FROM STACK AND SET RESPONSE TO DELEGATOR ID
509
SEND RESPONSE TO OPERATING SYSTEM
END

FIG. 5

ANNEX A TO THE DRAWINGS

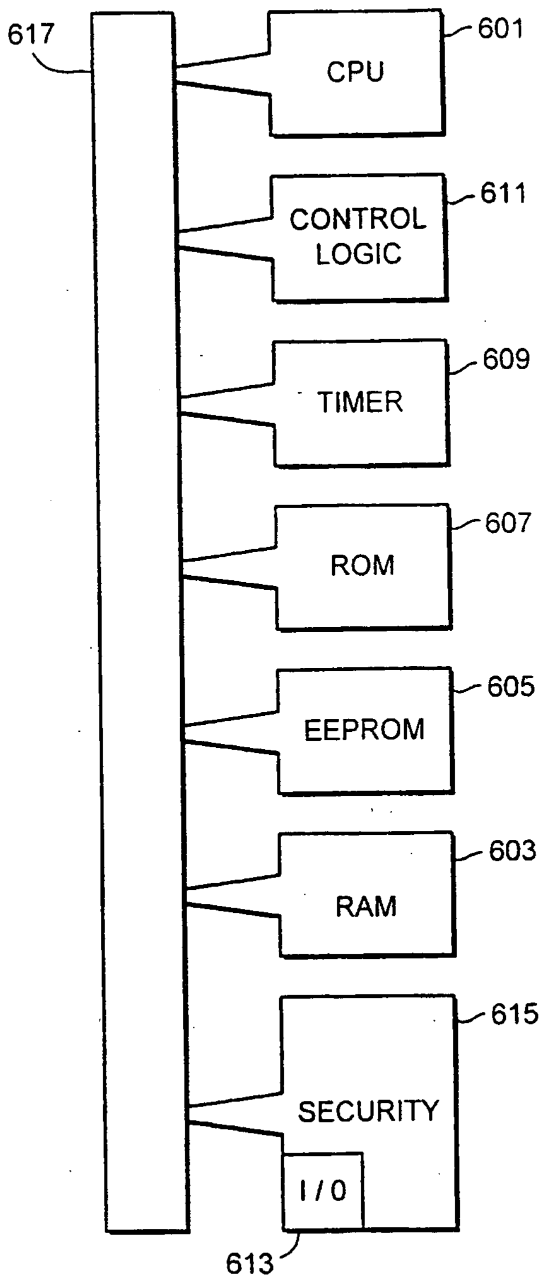


FIG. 6

ANNEX A TO THE DRAWINGS

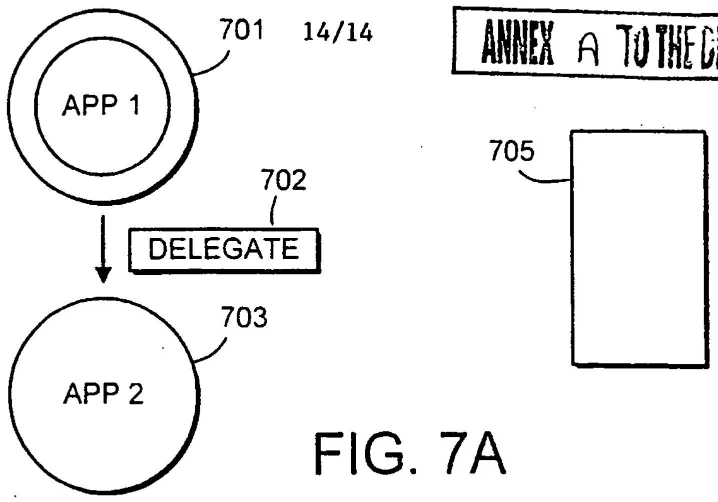


FIG. 7A

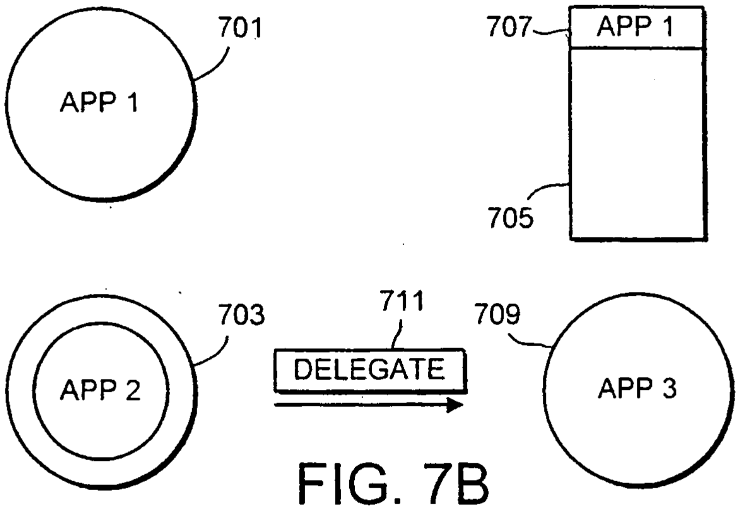


FIG. 7B

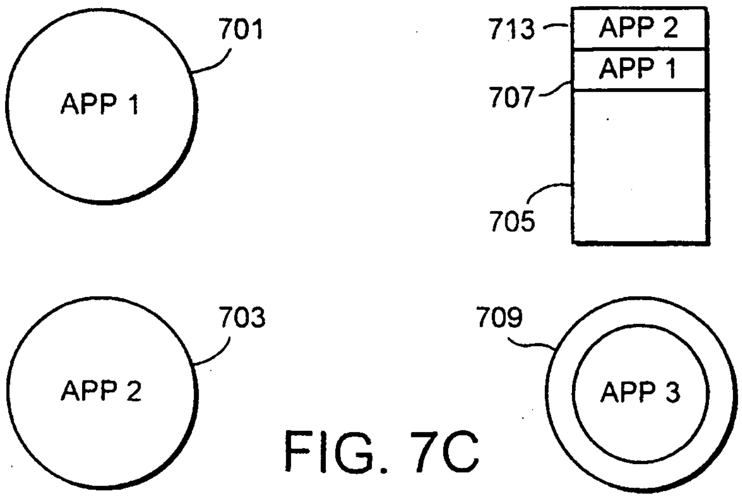


FIG. 7C